IBM AIX Parallel I/O File System: Installation, Administration, and Use

Document Number SH34-6065-02

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

Second Edition (November 1995)

This edition, SH34-6065-02, is a major revision of SH34-6065-01, which is now obsolete. Significant changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change. For a summary of the changes, see “What's New in this Book” on page xix.

This edition applies to the IBM AIX Parallel I/O FIle System Version 1 Release 1 and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for your comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX: (International Access Code)+1+914+432-9405

IBMLink (United States customers only): KGNRCF at KGNVMC
IBM Mail Exchange: USIB27BQ at IBMMAIL
Internet: kgnrcf@vnet.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Copyright International Business Machines Corporation 1995. All rights reserved. Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices xi
Trademarks xi

About This Book xiii
Who Should Use This Book xiii
How This Book Is Organized xiii
Typographic Conventions xiv
Related Publications xv
SP Hardware Publications xv
SP Software Publications xv
Recoverable Virtual Shared Disk Publications xv
LoadLeveler Publications xv
Parallel Environment for AIX Publications xvi
PVMe for AIX Publications xvi
Client Input Output/Sockets (CLIO/S) Publications xvi
Parallel I/O File System Publications xvi
Other IBM Publications xvi
AIX and RISC System/6000 Publications xvi

What's New in this Book xix

Overview of the IBM AIX Parallel I/O File System 1

Chapter 1. Overview of the Parallel I/O File System 3

Installing the Parallel I/O File System 7

Chapter 2. Requirements for the Parallel I/O File System 9
Hardware Requirements 9
Software Requirements 9
Installation Disk Space Requirements 9
Other Requirements 10
Program Material Needs 10
Licensing Information 10

Chapter 3. Installing and Configuring the Parallel I/O File System 13
Step 1: Install the Distribution Tape 13
Step 2: Install iFOR/LS (for AIX Version 3.2.5 only) 16
Step 3: Verify the Parallel I/O File System Installation 17
Step 4: Create the Configuration File 17
Step 5: Set Network Options 21
Step 6: Configure the Parallel I/O File System 21
Step 7: Format the Servers 22
Step 8: Start the Servers 22
Step 9: Mount the Parallel I/O File System on All the Client Nodes 23
Step 10: Authorize Users and Create User Directories 24
Step 11: Set Environment Variable for When No License is Available 24
Step 12: Set Up Libraries for Compiling and Linking 25
What To Do Next 25

© Copyright IBM Corp. 1995

Managing the Parallel I/O File System 27

Chapter 4. Administering the Parallel I/O File System 29
Starting a Parallel I/O File System 29
Stopping a Parallel I/O File System 29
Mounting a Parallel I/O File System 29
Unmounting a Parallel I/O File System 30
Rebooting the System 30
Managing User Authorization and Access 30
Determining the Status of a Parallel I/O File System Server 31
Unconfiguring a Parallel I/O File System 31
Removing a Parallel I/O File System 32
Changing Your Configuration 32
Adding or Deleting a Disk or Node 33
Handling License Availability 33
Finding Information about Errors 34
Troubleshooting PIOFS Problems 34
Getting Help From IBM 36

Chapter 5. Restrictions, Limitations, and Deviations 37
Risk to Permanent or Mission-Critical Files 37
PIOFS Restrictions, Limitations, and Deviations 37
Connection Limitations 38
Limits 38

Chapter 6. Parallel I/O File System Commands 39
add_piofs_user 40
change_piofs_user 41
delete_piofs_user 42
format_piofs_server 43
installpiofs 44
installpmp 45
list_piofs_users 46
piofs_config 47
piofs_fsck 49
piofs_verify 51
start_piofs_server 52
stat_piofs_server 53
stat_piofs_user 55
stop_piofs_server 57

Using the Parallel I/O File System 59

Chapter 7. Nuts and Bolts of Using PIOFS 61
Why Use Dynamic File Partitioning? 61
Overview of Dynamic File Partitioning 62
Useful Applications for a Parallel I/O File System 64
Understanding File Partitioning 65
Using PIOFS and AIX Functions to Create and Partition Files 71
Simple, Effective Ways to Partition a PIOFS File 72

Chapter 8. Programming Considerations 85

Understanding Atomicity of Reads and Writes . . . 85
Controlling Concurrent Access to Parallel I/O File System Files . . . 85
Maximizing PIOFS I/O Performance . . . 88
Checkpointing Files . . . 89
Specifying File Attributes for Parallel I/O File System Files . . . 89
Accessing PIOFS Files Correctly . . . 92
Using Existing Programs with a PIOFS File . . . 94

Chapter 9. Using PIOFS Functions and Base AIX Functions . . . 95
Using PIOFS Functions . . . 95
Supported AIX Functions . . . 97

Parallel I/O File System Programming Reference . . . 99
Programming Information for PIOFS Functions . . . 100
piofs_change_view . . . 101
piofs_chkpoint . . . 105
piofs_create . . . 107
piofs_delete_chkpoint . . . 110
piofs_fstat . . . 112
piofs_lseek . . . 115
piofs_restore_chkpoint . . . 118
piofs_statfs . . . 120

AIX Subroutines with Parallel I/O File System Considerations . . . 123
AIX Subroutine Documentation Conventions. . . . 124
Error Codes for Services that Require Path Name Resolution . . . 125
access, accessx, or faccessx . . . 126
chmod or fchmod . . . 129
chown, fchown, chownx, or fchownx . . . 133
close . . . 137
fclose or fflush . . . 139
fcntl, dup2, or dup . . . 141
fopen, freopen, or fdopen . . . 149
fread or fwrite . . . 154
fseek, rewind, ftell, fgetpos, or fsetpos . . . 157
getc, getchar, fgetc, or getw . . . 161
llseek (AIX Version 4.1) . . . 164
lseek . . . 166
mkdir . . . 168
open, openx, creat . . . 171
putc, putchar, fputc, or putw . . . 179
puts or fputs . . . 182
read, readx, readv, or readvx . . . 184
remove . . . 190
rename . . . 191
rmdir . . . 194
stat, lstat, fstat, statx, fstatx, fullstat, or ffullstat . . . 197
unlink . . . 202
write, writex, writev, or writevx . . . 204

Appendixes . . . 211

Appendix A. Parallel I/O File System Messages 213

Index 237

Figures

1. An RS/6000 SP with several client nodes accessing data at server nodes 4
2. A File of Numbers 62
3. A PIOFS file partitioned by column 63
4. A PIOFS file partitioned by row 63
5. A PIOFS file partitioned by column using three subfiles 63
6. A PIOFS file partitioned using a 3 x 2 block pattern 63
7. Steps involved in performing a Shear sort using a PIOFS file. 64
8. A matrix multiplication using partitioned PIOFS files 65
9. A simple PIOFS file: 4 cells spread across 2 server nodes. 66
10. A logical representation of a PIOFS File. 67
11. A PIOFS file being divided into subfiles 67
12. PIOFS file partitioning parameters and how they fit together 69
13. Logical order of BSUs in a block 70
14. Logical order of blocks in a subfile 70
15. A PIOFS File divided into 4 subfiles with a ghost cell (cell 11). 71
16. A PIOFS file partitioned by column - one cell per subfile 72
17. A PIOFS file partitioned by row 74
18. A Shear sort using a partitioned PIOFS file. 77
19. A matrix multiplication using partitioned PIOFS files 80
20. Write operations with no program control over the order of writes 87
21. Write operations with program control over the order of writes 87

© Copyright IBM Corp. 1995

Tables

1. Setting Up the Compiling and Linking Environment for C and C++ Programs 25
2. Setting Up the Compiling and Linking Environment for FORTRAN Programs 25
3. Parallel I/O File System Limits 38
4. Correct Ways to Establish Access to Files 2GB or Larger 93
5. Parallel I/O File System PIOFS C Language Functions 96
6. Parallel I/O File System FORTRAN Language Functions 97
7. AIX Subroutines Affected by the Parallel I/O File System 98

© Copyright IBM Corp. 1995

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P131
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
IBM
IBM AIX Parallel I/O File System
PVMe
AIX Parallel Environment
AIX Parallel System Support Programs
RISC System/6000
RISC System/6000 Scalable POWERparallel Systems

© Copyright IBM Corp. 1995

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This book explains how to install, manage, and use the IBM AIX Parallel I/O File System. In addition, it contains a listing of the messages generated by the IBM AIX Parallel I/O File System.

The information presented in this book assumes that you have a basic understanding of AIX file system structure.

Who Should Use This Book

This book is written for:

- AIX system administrators who are installing and maintaining the Parallel I/O File System
- Technical professionals who wish to modify or create applications to use the Parallel I/O File System
- Technical professionals interested in learning about the Parallel I/O File System.

How This Book Is Organized

The book is divided into the following parts:

- "Overview of the IBM AIX Parallel I/O File System" on page 1 describes the benefits and features of the Parallel I/O File System.
- "Installing the Parallel I/O File System" on page 7 lists the installation requirements and contains step-by-step installation instructions.
- "Managing the Parallel I/O File System" on page 27 describes how to start and stop the Parallel I/O File System, and how to control user access. PIOFS commands for system administration are also listed.
- "Using the Parallel I/O File System" on page 59 explains how to perform PIOFS file partitioning and provides coded examples of programs that use file partitioning. Programming considerations relating the use of PIOFS are also covered.
- "Parallel I/O File System Programming Reference" on page 99 lists the Parallel I/O File System programming functions available.
- "AIX Subroutines with Parallel I/O File System Considerations" on page 123 describes AIX subroutines whose functions are changed when used in Parallel I/O File System programs.
- Appendix A, "Parallel I/O File System Messages" on page 213 lists the Parallel I/O File System messages.

© Copyright IBM Corp. 1995

Typographic Conventions

This book uses the following typographic conventions:

Typographic	Usage
Bold	• **Bold** words or characters represent system elements that you must use literally, such as commands, flags, and path names. • **Bold** words also indicate the first use of a term included in the glossary.
Italic	• *Italic* words or characters represent variable values that you must supply. • *Italics* are also used for book titles and for general emphasis in text.
`Constant width`	Examples and information that the system displays appear in `constant width` typeface.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices. (In other words, it means “or.”)
< >	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <**Enter**> refers to the key on your terminal or workstation that is labeled with the word Enter.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
<**Ctrl-***x*>	The notation <**Ctrl-***x*> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>.

Related Publications

SP Hardware Publications

- *IBM RISC System/6000 Scalable POWERparallel Systems: Site Planning*, GC23-3905-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Maintenance Information, Volume 1, Installation and CE Operations*, GC23-3903-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Maintenance Information, Volume 2, Maintenance Analysis Procedures and Parts Catalog*, GC23-3904-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Library Guide*, GC23-3868-01
- *IBM RISC System/6000 Scalable POWERparallel Systems: Installation Aid Licensed Program Specification*, GC23-3838-02

SP Software Publications

- *IBM RISC System/6000 Scalable POWERparallel Systems: System Planning*, GC23-3902-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Administration Guide*, GC23-3897-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Installation Guide*, GC23-3898-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Diagnosis and Messages Guide*, GC23-3899-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: Command and Technical Reference*, GC23-3900-00
- *IBM RISC System/6000 Scalable POWERparallel Systems: IBM Parallel System Support Programs for AIX Licensed Program Specification*, GC23-3901-00

As an alternative to ordering the individual books, you can use GBOF-8587 to order the entire SP software library.

Recoverable Virtual Shared Disk Publications

- *IBM Recoverable Virtual Shared Disk: User's Guide and Reference*, GC23-3849-01
- *IBM Recoverable Virtual Shared Disk: Licensed Program Specification*, GC23-3848-01

LoadLeveler Publications

- *IBM LoadLeveler: Administration Guide*, SH26-7220-03
- *IBM LoadLeveler: User's Guide*, SH26-7226-03
- *IBM LoadLeveler: Licensed Program Specification*, GH23-0040-02

Parallel Environment for AIX Publications

- *IBM Parallel Environment for AIX: General Information*, GC23-3906-00
- *IBM Parallel Environment for AIX: Hitchhiker's Guide*, GC23-3895-00
- *IBM Parallel Environment for AIX: Installation, Administration, and Diagnosis*, GC23-3892-00
- *IBM Parallel Environment for AIX: Operation and Use*, GC23-3891-00
- *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*, GC23-3893-00
- *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*, GC23-3894-00
- *IBM Parallel Environment for AIX: Licensed Program Specification*, GC23-3896-00

As an alternative to ordering the individual books, you can use GBOF-8588 to order the entire Parallel Environment for AIX library.

PVMe for AIX Publications

- *IBM PVMe for AIX: Fact Sheet*, GC23-3907-00
- *IBM PVMe for AIX: User's Guide and Reference*, GC23-3884-00
- *IBM PVMe for AIX: Licensed Program Specification*, GC23-3885-00

Client Input Output/Sockets (CLIO/S) Publications

- *IBM CLIO/S General Information*, GC23-3879-00
- *IBM CLIO/S Licensed Program Specification*, GC23-3789-01
- *IBM CLIO/S User's Guide and Reference*, available from your IBM representative

Parallel I/O File System Publications

- *IBM AIX Parallel I/O File System: Fact Sheet*, G325-0649-02
- *IBM AIX Parallel I/O File System: Licensed Program Specification*, GH34-6066-01

Other IBM Publications

AIX and RISC System/6000 Publications

- *IBM AIX for RISC System/6000 System Overview*, GC23-2406
- *IBM Getting Started: Using IBM RISC System/6000*, GC23-2377
- *IBM Getting Started: Managing RISC System/6000*, GC23-2378
- *IBM General Concepts and Procedures for RISC System/6000*, GC23-2202
- *IBM AIX Communication Concepts and Procedures*, GBOF-1524
- *IBM AIX Version 3.2 for RISC System/6000 Files Reference*, GC23-2200
- *IBM AIX Version 4.1 Files Reference*, SC23-2512

- *IBM AIX Version 3.2 System Management Guide*, SC23-2457
- *IBM AIX Version 3.2 System Management Guide: Communications and Networks*, GC23-2487
- *IBM AIX Version 4.1 System Management Guide: Communications and Networks*, SC23-2526
- *IBM AIX Version 3.2 Installation Guide*, SC23-2341
- *IBM AIX Version 4.1 Installation Guide*, SC23-2550
- *IBM AIX for RISC System/6000 Commands Reference*, GBOF-1802
- *IBM AIX Version 4.1 Commands Reference*, SBOF-1851
- *IBM AIX Versions 3.2 and 4.1 Performance Tuning Guide*, SC23-2365
- *IBM RISC System/6000 Problem Solving Guide*, SC23-2204
- *IBM AIX Version 4.1 Problem Solving Guide and Reference*, SC23-2606
- *IBM AIX Version 4.1 Messages Guide and Reference*, SC23-2641
- *IBM AIX Version 4.1 Network Installation Management Guide and Reference*, SC23-2627
- *IBM AIX Version 4.1 System Management Guide: Operating System and Devices*, SC23-2525
- *IBM AIX Version 4.1 General Programming Concepts: Writing and Debugging Programs*, SC23-2533
- *IBM AIX Version 4.1 Communications Programming Concepts* SC23-2610
- *Diskless Workstation Management Guide*, SC23-2433
- *C++ for AIX/6000: Language Reference*, SC09-1606
- *C++ for AIX/6000: Standard Class Library Reference*, SC09-1604
- *C++ for AIX/6000: User's Guide*, SC09-1605
- *C++ for AIX/6000: Language Reference*, SC09-1606
- *C++ for AIX/6000: Standard Class Library Reference*, SC09-1604
- *C++ for AIX/6000: User's Guide*, SC09-1605
- *Parallel ESSL Guide and Reference*, GC23-3836
- *ESSL Version 2 Release 2 Guide and Reference*, SC23-0526
- *ESSL Products General Information*, GC23-0529

What's New in this Book

Summary of Changes for SH34-6065-02

This major revision contains changes to support the Parallel I/O File System Version 1 Release 1 with PIOFS PTF numbers U440596–U440599 installed.

Changed Information

- PIOFS for AIX Version 3.2.5 now enables FORTRAN programs to access files 2GB or larger.
- The software requirements information includes a requirement for new FORTRAN PTFs.

This revision also includes technical and editorial improvements. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

© Copyright IBM Corp. 1995

Overview of the IBM AIX Parallel I/O File System

© Copyright IBM Corp. 1995

Chapter 1. Overview of the Parallel I/O File System

The Parallel I/O File System (PIOFS) is designed for serial or parallel applications that require large temporary files and high I/O bandwidth.

PIOFS lets you create files as large as 128 Terabytes that span multiple server nodes. A PIOFS file can be treated as a normal AIX file or it can be logically partitioned into subfiles, each containing a portion of the file's data. Each subfile can then be processed in parallel by a separate task to let you perform large computational tasks faster and more efficiently.

With PIOFS file partitioning, you can parallelize access to your data without the inconvenience and administrative overhead of maintaining multiple data files. PIOFS files can be dynamically partitioned into subfiles many different ways, all without altering or moving the contents of the file.

PIOFS supports parallelism in two complementary ways: physically and logically:

- A file can be divided physically over multiple disks and servers.
- A file can be divided logically into multiple subfiles.

Both the physical and logical division of the file contribute to the ability to process the file in parallel.

Another key benefit of PIOFS is its scalability. Large files can be processed at the same speed as smaller files; you simply divide them into a larger number of subfiles and create additional processes.

PIOFS lets you increase the parallelism of many of your existing C and FORTRAN applications with little or no modification of the code. You can even achieve significant parallelism for your application just by using PIOFS to spread files across multiple disks and servers, without using file partitioning.

You can also write new applications to take the greatest advantage of the Parallel I/O File System's capabilities. PIOFS functions let you implement complex algorithms with less complicated programming.

This chapter gives you an overview of the Parallel I/O File System and its features. The details of how to create and partition PIOFS files are presented in Chapter 7, “Nuts and Bolts of Using PIOFS” on page 61.

Parallel File Access

A leading feature of the Parallel I/O File System is increased efficiency of file access. The RS/6000 SP lets you divide your application's compute tasks among a number of client nodes. Your application's tasks might each execute at the optimum speed of the client node. But, without parallel data access, one task often waits for access to the server node while another task reads or writes data. The Parallel I/O File System changes that.

Imagine your RS/6000 SP as four client nodes connected through a network (the SP High Performance Switch provides the most efficient access) to four server nodes. Your installation can actually have from four nodes up to the full complement provided by the RS/6000 SP.

© Copyright IBM Corp. 1995

Now imagine you have a parallel application running on those client nodes. With the Parallel I/O File System, you can have a single file spread across the four server nodes. Any one of the client nodes can access data at any of the server nodes as shown in Figure 1 on page 4.

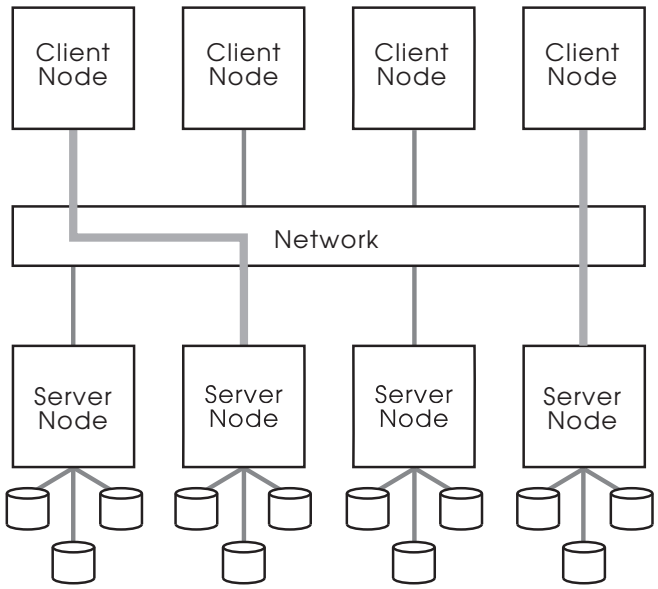

Figure 1. An RS/6000 SP with several client nodes accessing data at server nodes. The Parallel I/O File System supports simultaneous access of server nodes by multiple client nodes.

Because multiple client nodes can access different parts of a file in parallel, instead of serially, tasks on client nodes no longer have to wait their turn to access a shared file. Therefore, your application can access its data more quickly.

In addition, you can logically partition the file into multiple disjoint subfiles and have each subfile processed by a different task, further increasing the parallelism of your application.

To achieve maximum parallelism, you can structure your PIOFS file so that each subfile is physically located on a different server node and is processed by a different task.

Up To 128-Terabyte File Support

With its own file offset function, the Parallel I/O File System supports files up to 128 Terabytes in size. The upper limit, subject to the *block_limit* set in your configuration file, is the combined space of all your disks.

The Parallel I/O File System is architected to support a file size of up to 2^{63} bytes. The currently implemented limit, however, is a file size of (512 nodes x 256GB per node) bytes.

Scalability

Because PIOFS lets you divide a file into disjoint subfiles, each to be processed in parallel by a different task, large files can be processed at the same speed as smaller files. As your files grow, your processing time need not increase. You can just divide the file into more subfiles and have additional tasks process them.

Use with Existing Applications

In some cases, you can use the Parallel I/O File System with an existing C or FORTRAN application with only minor changes required.

PIOFS files can be treated as standard AIX files or they can be divided into subfiles and processed in parallel by multiple tasks. An additional benefit for existing files: they can grow to the greater size supported by the Parallel I/O File System.

Portability and Application Support

The Parallel I/O File System's use of existing AIX read, write, open, and close functions lets many of your serial file access programs work unchanged. You can use a system-wide profile or create your own profile file that lets your existing programs take advantage of certain Parallel I/O File System features.

The Parallel I/O File System has both C and FORTRAN functions for creating, checkpointing, and returning information about parallel files and server nodes. See Chapter 8, “Programming Considerations” on page 85 for a list of the PIOFS C and FORTRAN functions. That chapter also lists changed and unsupported AIX functions.

Creating and Copying Files

Files smaller than 2GB can be copied between a Parallel I/O File System and another file system. You can also create new files.

Checkpointing

The PIOFS file checkpointing function (not to be confused with application checkpointing) lets you create a snapshot of a PIOFS file's contents at a particular point during processing. If your program fails, you can use the checkpointed version of the file to restart the program where it was last checkpointed. You can also use file checkpointing to save a particular version of a PIOFS file.

You can read the checkpointed version of a file while the active version is being accessed.

Coexistence With Other Parallel Processing Products

The Parallel I/O File System can coexist with other products that support parallel processing on the Scalable POWERparallel Systems, such as:

- IBM AIX Parallel Environment
- Express
- Original Linda
- PVM
- PVMe

Installing the Parallel I/O File System

Chapter 2. Requirements for the Parallel I/O File System 9
Hardware Requirements 9
Software Requirements 9
Installation Disk Space Requirements 9
Other Requirements 10
Program Material Needs 10
Licensing Information 10

Chapter 3. Installing and Configuring the Parallel I/O File System 13
Step 1: Install the Distribution Tape 13
Step 2: Install iFOR/LS (for AIX Version 3.2.5 only) 16
Step 3: Verify the Parallel I/O File System Installation 17
Step 4: Create the Configuration File 17
Step 5: Set Network Options 21
Step 6: Configure the Parallel I/O File System 21
Step 7: Format the Servers 22
Step 8: Start the Servers 22
Step 9: Mount the Parallel I/O File System on All the Client Nodes 23
Step 10: Authorize Users and Create User Directories 24
Step 11: Set Environment Variable for When No License is Available 24
Step 12: Set Up Libraries for Compiling and Linking 25
What To Do Next 25

© Copyright IBM Corp. 1995

Chapter 2. Requirements for the Parallel I/O File System

This chapter lists the requirements for installing, configuring, and using a Parallel I/O File System.

Hardware Requirements

The following hardware is required:

- IBM RS/6000 Scalable POWERparallel Systems

Although the SP High Performance Switch is optional, using it will help you achieve maximum performance.

Software Requirements

For the Parallel I/O File System for AIX Version 3.2.5:

- IBM AIX/6000 Version 3 Release 2.5 (5756-030) operating system and the IBM AIX Parallel System Support Programs (PSSP) Version 1 Release 2 (5765-296), which are included with IBM RS/6000 Scalable POWERparallel Systems
- If FORTRAN will be used, AIX XL FORTRAN Compiler/6000 Version 3 Release 2 (5765-176) or later with either:
  - PTF U440137, if you plan only to run FORTRAN programs
  - PTF U440138, if you plan to compile and run FORTRAN programs.
- iFOR/LS for licensing, which is shipped on the Parallel I/O File System distribution tape

For the Parallel I/O File System for AIX Version 4.1:

- IBM AIX Version 4 Release 1.3 (5765-393) operating system and the IBM AIX Parallel System Support Programs (PSSP) Version 2 Release 1 (5765-529), which are included with IBM RS/6000 Scalable POWERparallel Systems
- If FORTRAN will be used, AIX XL FORTRAN Compiler/6000 Version 3 Release 2 (5765-176) or later with either:
  - PTF U440135, if you plan only to run FORTRAN programs
  - PTF U440136, if you plan to compile and run FORTRAN programs.
- If the PIOFS C functions will be used, the IBM C FOR AIX Version 3.1 (5765-423) or later is required (the C compiler is no longer shipped as part of the AIX program product).

Installation Disk Space Requirements

The Parallel I/O File System requires the following amounts of space in **/usr/lpp/piofs**.

© Copyright IBM Corp. 1995

Usage	Space Required
Parallel I/O File System	10 Megabytes
iFOR/LS	6 Megabytes to install an iFOR/LS license server (for AIX Version 3.2.5 only)
	4K to install nodelocked licenses
:pk/var/piofs	12K for error logs

Other Requirements

Your RS/6000 SP and the software required to support the Parallel I/O File System must be installed and operational.

The root id must be able to **rsh** to all the client and server nodes without specifying a password.

To install the Parallel I/O File System, you need to perform all of the installation tasks as root.

Program Material Needs

Make sure you have the following items before you start the installation process:

1. The distribution tape for either the AIX Version 3.2.5 or AIX Version 4.1 version of the Parallel I/O File System

   The distribution tape contains a **README** file (**/usr/lpp/piofs/README/piofs.README**), that describes last minute changes and any required PTFs. Read the **README** file before you start the Parallel I/O File System installation process.

2. iFOR/LS and its README file in **/usr/lib/netls**. (You may already have this program installed.)

Licensing Information

The Parallel I/O File System uses iFOR/LS to regulate license use. iFOR/LS was formerly called Net/LS, a name that still appears in some documentation and software, If you already have iFOR/LS installed, you do not have to reinstall it. You simply need to order your NetLS password.

Only servers are licensed. You need one license for each server node. Client nodes do not require licenses.

If you do not have NetLS installed, you will have to install the NetLS server version provided with PIOFS. For information on installing NetLS and acquiring your NetLS password, refer to “Step 2: Install iFOR/LS (for AIX Version 3.2.5 only)” on page 16.

The Parallel I/O File System installation tape contains the directory: **/usr/lib/netls** with the iFOR/LS license server code and documentation in it.

You should be familiar with the following:

- *NetLS Quick Start Guide*, SC09-1661

- *Managing Software Development with Network Licensing System*, SC09-1660
- *Managing NCS Software*, SC09-1834

Chapter 3. Installing and Configuring the Parallel I/O File System

This chapter walks you through the Parallel I/O File System installation and configuration process, which involves the following steps:

"Step 1: Install the Distribution Tape"

"Step 2: Install iFOR/LS (for AIX Version 3.2.5 only)" on page 16

"Step 3: Verify the Parallel I/O File System Installation" on page 17

"Step 4: Create the Configuration File" on page 17

"Step 5: Set Network Options" on page 21

"Step 6: Configure the Parallel I/O File System" on page 21

"Step 7: Format the Servers" on page 22

"Step 8: Start the Servers" on page 22

"Step 9: Mount the Parallel I/O File System on All the Client Nodes" on page 23

"Step 10: Authorize Users and Create User Directories" on page 24

"Step 11: Set Environment Variable for When No License is Available" on page 24

"Step 12: Set Up Libraries for Compiling and Linking" on page 25

Step 1: Install the Distribution Tape

Install the Parallel I/O File System on the client nodes and the licensed server nodes of your RS/6000 SP, using the **installp** process. The Parallel I/O File System installp image consists of several components:

.

1. **The client component** — Provides only the functions for PIOFS clients.
2. **The server component** — Provides the functions for PIOFS servers. The server component also includes the README file as well as the shell scripts used to administer PIOFS.
3. **The common component** — Provides functions used by both the client and server parts of PIOFS. The common component is automatically installed when you install either the client or server component.
4. **The SMIT screens** — Allow you to use SMIT for PIOFS-related system management tasks. The SMIT screens require that the server component also be installed.

A separate installp image, **piofs.docs** contains the PIOFS documentation:

- A softcopy file of this book, *IBM AIX Parallel I/O File System: Installation, Administration, and Use*, formatted for PostScript printing
- The InfoExplorer database for PIOFS
- PIOFS man pages.

© Copyright IBM Corp. 1995

"Finding and Using the Documentation" on page 14 provides additional information about PIOFS documentation.

You can install the PIOFS documentation on any node, including those with no PIOFS functions installed. You can use the **dsh** command to install the images on the other client and server nodes.

Note

Be sure to read the PIOFS README file before installing PIOFS. The README file contains important performance and tuning tips as well as other necessary information.

The README file is in **/usr/lpp/piofs/README/piofs.README**.

See "Step 2: Install iFOR/LS (for AIX Version 3.2.5 only)" on page 16 if you have any questions about licensed nodes. See *IBM 9076 Scalable POWERparallel Systems: SP Installation Guide* for general installation instructions for the RS/6000 SP.

Selecting the Nodes on Which to Install Parallel I/O File Systems

Programs containing calls to PIOFS functions can be loaded only on nodes that have PIOFS installed and mounted. This is necessary because all function calls must be resolved regardless of whether they will be executed.

Note: You can configure multiple Parallel I/O File Systems, but you can have only one Parallel I/O File System on each set of server nodes.

Ensuring Proper Installation

You can check to see that the Parallel I/O File System has been installed with the AIX command **lslpp**:

```
lslpp -h piofs.*
```

Finding and Using the Documentation

The Parallel I/O File System provides the following documentation:

- This document, *IBM AIX Parallel I/O File System: Installation, Administration, and Use*, which you can browse online or read hardcopy.
- Man pages — online help on AIX workstations
- A PIOFS InfoExplorer database.

The Parallel I/O File System Program Directory describes how to install these PIOFS documentation components on your workstation.

As mentioned previously, the PIOFS documentation is contained in a separate installp image, **piofs.doc**. The README file, however, is in the server component of the PIOFS installp image.

IBM AIX Parallel I/O File System: Installation, Administration, and Use

You can obtain *IBM AIX Parallel I/O File System: Installation, Administration, and Use* in the following ways:

- A file formatted for PostScript previewing or printing is available on AIX in the file **/usr/lpp/piofs/docs/UsersGuide.ps**.
- A PostScript file of the latest version of the book is available from the anonymous FTP server **lscftp.kgn.ibm.com** (129.40.10.220). The file name is **/pub/pps/piofs/UsersGuide.ps.Z**.
- A printed copy of the book is available from the IBM Distribution Center.

Downloading this Book from the Anonymous FTP Server: To obtain this book from the anonymous FTP server:

1. Use **ftp** to connect to **lscftp.kgn.ibm.com** (129.40.10.220) using the user ID `anonymous`. Use your Internet address as the password.
2. Download file **pub/pps/piofs/UsersGuide.ps.Z** in binary mode. The file is compressed.
3. Uncompress it using the AIX **uncompress** command.
4. Print it as you print other PostScript files.

PIOFS Man Pages

AIX man pages, or online help, are provided for the Parallel I/O File System commands and subroutines. The man page files are in **/usr/lpp/piofs/man**.

- The man pages for functions are installed in **/usr/lpp/piofs/man/cat3**.
- The man page for the **piofs.profile** file is installed in **/usr/lpp/piofs/man/cat4**.
- The man pages for commands are installed in **/usr/lpp/piofs/man/cat8**.

The man pages are in the **/usr/lpp/piofs/man** structure. Because this is not one of the directories automatically read by the AIX **man** command, use one of these ways to indicate which man page directories should be searched:

- Specify the –M*manpath* flag when you invoke the **man** command.
  For example, you can display the Parallel I/O File System man pages using

  ```
  man -M/usr/lpp/piofs/man  topic
  ```

- Set the MANPATH environment variable to the top of the man page structure. Consult the reference information for the AIX operating system for information about setting MANPATH.

Tips on Using Man Pages: To view a Parallel I/O File System man page, specify by name any of the Parallel I/O File System commands or subroutines when you invoke the **man** command. For example, `man start_piofs_server` displays the man page for the **start_piofs_server** command. Man page names are case sensitive. Specify the name in the case shown in this manual.

For more efficient information retrieval, include the number of the man page section when you issue the **man** command. For instance, to view the man pages for the Parallel I/O File System subroutines, which are in section three of the manual, issue:

```
man 3 piofs_change_view
```

The PIOFS InfoExplorer Database

The InfoExplorer database files are in **/usr/lpp/piofs/info**. To bring up InfoExplorer to browse the PIOFS documentation, type:

```
info -l /usr/lpp/piofs/info
```

Step 2: Install iFOR/LS (for AIX Version 3.2.5 only)

After installing the Parallel I/O File System, the **/usr/lpp/piofs/netls** directory contains the following files:

- README.FIRST
- README.gradient
- netlsark.tar.Z

Read the README files if you plan to use the iFOR/ls supplied with the Parallel I/O File System. The file, netlsark.tar.Z, is a packed file containing the iFOR/ls program. To unpack the file, type:

cd /usr/lpp/piofs/netls;
uncompress netlsark.tar.Z;
tar -xlf netlsark.tar

Note that after installation is complete, you can delete the **netlsark.tar** file.

Now the AIX directory contains everything that you need to install and configure iFOR/LS. This includes the following:

- Installation instructions in **/usr/lpp/piofs/netls/AIX/README**
- **netlsd** daemon
- Associated administrative commands
- Associated man pages
- Network Computing System (NCS).

The AIX version of iFOR/LS is supported on the IBM RISC System/6000 running AIX 3.2.5. The iFOR/LS provided may work with other versions of AIX, however, these other versions are not supported. Administration commands for iFOR/LS support both command line and Motif interfaces. The graphical user interface requires that X11R5 be installed on the system.

The complete iFOR/LS package for AIX 3.2.5 consists of an **installp** image called netlsArk for RISC System/6000 workstations.

The System Management Interface Tool (SMIT) procedure provides an automated method of installing only those components required in a given situation.

The NCS component provides the Remote Procedure Call libraries along with the local and global naming services that are needed to correctly operate iFOR/LS in a network environment. The netlsARK lets you administer a network of license servers and provides a license server on the local machine.

If you intend to install iFOR/LS on a system that is already running applications licensed with the Resource License Manager, be aware that iFOR/LS can coexist in the same network partition as the Resource License Manager. Additionally, Resource License Manager licensed client applications can have their license

requests satisfied by the iFOR/LS server, provided that the iFOR/LS server is in the default cell. If you encounter problems, contact IBM for assistance.

Step 3: Verify the Parallel I/O File System Installation

When you have completed installation, but before you configure the Parallel I/O File System, you can use the **piofs_verify** command to check out your installation on each node. The command checks for any required programs, defines and creates a small Parallel I/O File System file, then removes the file. It tells you if it finds any problems. You must be **root** to run this program. To run the verification program issue the command:

```
/usr/lpp/piofs/bin/piofs_verify
```

See "piofs_verify" on page 51 for the syntax of the **piofs_verify** command.

Step 4: Create the Configuration File

Files stored in a Parallel I/O File System are distributed across a set of server nodes, each having one or more disks. Before initializing a Parallel I/O File System, you need to specify the server nodes and their accompanying disks by configuring the Parallel I/O File System.

Planning Logical Volume Space Allocation

It is best to configure all the server nodes with about the same total amount of logical volume space. A Parallel I/O File System uses about the same amount of logical volume space on each server. For instance, if the logical volume sizes of server node A add up to 100 MB and the logical volume sizes of server node B add up to 80 MB, the Parallel I/O File System will use only approximately 80 MB on server node A. If the space allocated to the Parallel I/O File System at one server node is full, the Parallel I/O File System returns an out of space message even if space is available at other nodes.

If you are creating multiple Parallel I/O File Systems, you will need a configuration file for each instance. File systems are independent of one another. Each server node can have only a single Parallel I/O File System, although a particular PIOFS can be associated with multiple server nodes.

After you create the configuration file, you can configure the Parallel I/O File System by using SMIT or by entering individual commands.

Specifying the Contents of the Configuration File

To create the configuration file either copy and edit the supplied sample configuration file or create your own.

The configuration file contains the addresses of the server nodes that will hold the Parallel I/O File System-defined files and specifies how much file space users are allowed.

The most straightforward way to create a configuration file is to copy the sample configuration file (called **sample.piofscfg**) provided with the Parallel I/O File System and name the copy **piofs.piofscfg**. You can give the file another name, but you must refer to that name later when you issue **piofs_config**. The

sample.piofscfg file is located in the /**usr**/**lpp**/**piofs** directory. To create a copy of this file, type:

```
cp  /usr/lpp/piofs/sample.piofscfg /var/piofs/config_file_name
```

Note that different configuration files must have different file names. Different path names are not sufficient to distinguish between identically-named configuration files. You should end all configuration files with **.piofscfg** because SMIT searches /**var**/**piofs** and /**usr**/**lpp**/**piofs** for file names ending with **.piofscfg**. when you request a list of configuration files.

Before editing your configuration file, familiarize yourself with the sample configuration file and its syntax.

Sample Configuration File

The sample, **sample.piofscfg**, includes all the required and optional lines. Your specific file may be longer.

```
#    This is a sample Configuration File
block_limit   nnn
object_limit  nnn
IPports       9076 9077
network_interface_name css0
udp_sendspace nnn
client_udp_recvspace nnn
server_udp_recvspace nnn
# Storage nodes are server nodes
storage_node 0  node1   000.000.000.1  1         100   rootvg   hdisk1
storage_node 0  node1   000.000.000.1  2         100   rootvg   hdisk2
storage_node 0  node1   000.000.000.1  metadata    7   rootvg   hdisk1
storage_node 1  node2   000.00.00.2    1         200   rootvg   hdisk1
storage_node 1  node2   000.00.00.2    metadata    7   rootvg   hdisk1
```

Configuration File Syntax

- Comments begin with a pound sign (#).
- Extra blanks and tabs in a line are ignored. Tokens must be separated with a space.
- The keywords are case sensitive. Be careful to get the case correct for the **IPports** keyword.

Configuration File Keywords And Syntax

block_limit `block_limit nnn`

where *nnn* is the number of 64K blocks per user per server node for all users at the site.

The number must be between 1 and 134217727. It specifies the number of 64K data blocks that any one user can have on a single server node. The default is to use as many blocks as are physically available. That amount could exceed 134217727.

To specify an unlimited number of blocks, either use -1 or omit the block limit line. To set the block limit to 65536 64K blocks per user per server node, specify:

```
block_limit 65536
```

object_limit

```
object_limit nnn
```

where nnn is the number of objects per user per server node.

An object is a file, a *cell* of a file, or a directory, used by the Parallel I/O File System.

This number must be between 0 and 134217727. You need at least one object per cell plus one for the file information. Therefore, for each file you need the following number of objects:

1 × *number_of_cells* + 1

The default is to use as many objects as will fit in the space available. That amount could exceed 134217727.

To use as many objects as will fit in the space available, specify -1. To set the object limit to 512 objects:

```
object_limit 512
```

IPports

```
IPports nnn mmm
```

where *nnn* and *mmm* are IP port numbers to use.

This keyword specifies the IP port numbers that will be used by the client and server nodes to communicate. The defaults are 9076 and 9077.

To specify server node IP ports 9076 and 9077:

```
IPports 9076 9077
```

network_interface_name

```
network_interface_name css0
```

This is the network interface name of the communications adapter that will be used for all the communications between the servers and between clients and servers. The default is css0, the SP High Performance Switch.

To set the network_interface_name to css0

```
network_interface_name css0
```

udp_sendspace

```
udp_sendspace nnn
```

The UDP socket send buffer size (in bytes) to be used by the Parallel I/O File System. The value of **udp_sendspace** determines the amount of data transferred at a time between client and server. The recommended value is 131072 bytes.

Note that if the **udp_sendspace** value is less than 65536, performance may suffer due to the larger volume of messages.

client_udp_recvspace

```
client_udp_recvspace nnn
```

server_udp_recvspace

`server_udp_recvspace` *nnn*

The UDP socket receive buffer sizes (in bytes) to be used by the Parallel I/O File System. We recommend that you assign the values using the following formulas:

```
client_udp_recvspace = 131072 (128K) * # of servers
server_udp_recvspace = 131072 (128K) * # of clients
```

For large systems, you might want to change the formulas by replacing:

# of clients — with the approximate number of clients active at any one time

# of servers — with the approximate number of servers active at any one time.

Note that if the UDP socket receive buffers are too small, messages will be dropped, requiring re-transmission with resulting performance degradation.

storage_node

`storage_node` *lnode hostname IPaddr devnum lvsize volgrp pdisk*

A storage node is a server node. The **storage_node** entries require one line for each logical volume assigned to a storage node:

lnode — Logical storage node number. The first number used must be 0; succeeding node numbers must be in sequential order. Each *hostname* entry needs a separate lnode entry.

hostname — Hostname returned by the **hostname** command when run at that server node.

IPaddr — IP address to use for client-server communication. This is the address of the server. If the SP High Performance Switch is used, this is the address of the switch.

Note that all the servers must use the same adapter types. You cannot have some nodes switch-attached and some ethernet-attached. For best performance, all nodes should be switch-attached.

devnum — Device number—the entities that the file system will stripe data across at that server node. Logical volumes with Journaled File System (JFS) file systems on them will be created using these numbers:

```
/dev/pfsd1    # device 1
/dev/pfsd2    # device 2
⋮
/dev/pfsmd    # metadata
```

The device numbers must start at 1 and increase sequentially to a maximum of 128 for each node. There must also be a device specified as **metadata** at each node which will hold the file system metadata.

lvsize — Size of the logical volume in physical partitions. (A physical partition is usually 4MB.)

The *lvsize* for metadata devices must be at least 28MB or seven 4MB physical partitions. The *lvsize* for all devices cannot exceed 2GB or 512 4MB physical partitions.

volgrp — Volume group in which to create the logical volume.

You can use existing volume groups or use SMIT to define new volume groups.

pdisk — Physical disk on which to create the logical volume. You can create several logical volumes on one physical disk but this prevents you from getting multiple disk heads active at the same time.

These entries create the underlying JFS file systems that the Parallel I/O File System will use.

For example:

```
storage_node 0  node1  123.45.678.90  1         5  rootvg  hdisk1
storage_node 0  node1  123.45.678.90  metadata  7  rootvg  hdisk1
```

Step 5: Set Network Options

Set the network options on each server as specified in the README file (**/usr/lpp/piofs/README/piofs.README**).

Step 6: Configure the Parallel I/O File System

Configuration creates logical volumes on each disk on each server node specified in the configuration file. It also verifies the syntax of the configuration file and makes copies of the configuration files for the servers.

You can configure a Parallel I/O File System, format the servers (step 7), and start the servers (step 8), all using the System Management Interface Tool (SMIT). Alternatively, you can issue commands to perform these steps. If you use SMIT, go to the Parallel I/O File System menu as explained below and use it to:

1. Configure the Parallel I/O File System
2. Format the servers
3. Start the servers.

Then, go directly to “Step 9: Mount the Parallel I/O File System on All the Client Nodes” on page 23. If you use the command interface, perform each step in sequence.

Using SMIT

To use SMIT, do either of the following:

- Use the fast path by typing **smit piofs**
- Type **smit** and pass through the sequence of menus listed below until you reach the one for the Parallel I/O File System:
  - Physical & Logical Storage
  - File Systems
  - Add / Change / Show /Delete File Systems
  - Parallel I/O File System

Using the piofs_config command

To configure the Parallel I/O File System, use the **piofs_config** command. Run this command as root from the same node where you created and edited the configuration file.

```
/usr/lpp/piofs/bin/piofs_config /var/piofs/config_file_name
```

Use the full path name for the configuration file when issuing the **piofs_config** command. See "piofs_config" on page 47 for a full description of the **piofs_config** command.

Step 7: Format the Servers

Formatting cleans up the PIOFS file space during the initial installation (or if you are reconfiguring). You do this step *only* after configuring a Parallel I/O File System. You do not do this step if you are restarting the Parallel I/O File System.

Caution

Formatting destroys all the PIOFS files on the target disk.

1. Login to one of the servers as root.
2. Type:

   ```
   /usr/lpp/piofs/bin/format_piofs_server config_file_name
   ```

   Specify only the configuration file name, *not* a full path name.

The **format_piofs_server** command is documented in "format_piofs_server" on page 43.

Step 8: Start the Servers

To start the servers:

1. Login to one of the server nodes as root.
2. Type:

   ```
   /usr/lpp/piofs/bin/start_piofs_server config_file_name
   ```

Specify only the configuration file name, not a full path name. If you have configured multiple Parallel I/O File Systems, each with a separate configuration file, be sure to use the correct configuration file name.

See “start_piofs_server” on page 52 for complete documentation of the **start_piofs_server** command.

Step 9: Mount the Parallel I/O File System on All the Client Nodes

To mount a Parallel I/O File System, login using the root id. This process loads the file system as a kernel extension on the first mount command.

If your RS/6000 SP has an active system data repository (SDR), you can omit sub-step 1 and go to sub-step 2.

The following sub-steps must be done on each client node.

1. Find the active configuration files in **/etc** on any configured server. The names are:

   /**etc**/*config_file_name***.active**

   /**etc**/*config_file_name***.IPaddresses.active**

   These are the configuration files you created with **.active** and **.IPaddresses.active** appended to it.

   Copy the active configuration files into the **/etc** directory on each of the clients.

2. Use the **crvfs** command to add the following line to **/etc/vfs**:

   ```
   piofs nnn   /usr/lpp/piofs/bin/piofsmnthelp none
   ```

   Replace *nnn* with a number greater than 8 and less than 12 that is not used for any other file system type at your installation.

   To do this, type:

   ```
   crvfs "piofs:nnn:/usr/lpp/piofs/bin/piofsmnthelp:none"
   ```

   Refer to the **vfs File** information for the AIX Base Operating System for a description of **/etc/vfs**.

3. Use the **crfs** command to add the following stanza to **/etc/filesystems** on all the client nodes where you ran **installp**:

   ```
    # You can specify, on the line below, another name
    # that the mount command will find.
   /your_mount_point:
    # You can replace piofs with any name on the line below
    dev         = /piofs
    # You can replace piofs with any name on the line below or omit
    # the next line entirely. It is optional.
    type        = piofs
    vfs         = piofs
    # Leave mount = false so that mounts are not performed automatically
    # when the system is booted
    mount       = false
    # Specify config_file_name without the full path name on the line below
    options     = config_file_name
    # You can replace piofs with any name on the line below
    nodename    = piofs
   ```

   To do this, type:

   ```
   crfs -v piofs -u piofs -d /piofs -n piofs -m /your_mount_point
   -a options=config_file_name
   ```

You can use the **chfs** command to change /**etc**/**filesystems** entries and **rmfs** to remove /**etc**/**filesystems** entries.

4. If you didn't use the **crfs** command to modify /**etc**/**filesystems**, make a mount point of */your_mount_point* by issuing:

   ```
   mkdir /your_mount_point
   ```

   This is the mount point you specified for the `dev` field.

5. Set the network options for this client node as specified in the README file (/**usr**/**lpp**/**piofs**/**README**/**piofs.README**).

6. Mount the file system. Using the entry in /**etc**/**filesystems** issue:

   ```
   mount /your_mount_point
   ```

   You could also enter the following from the command line;

   ```
   mount -n piofs -o config_file_name  -v piofs /piofs /your_mount_point
   ```

Step 10: Authorize Users and Create User Directories

Users other than root must be individually authorized to create files within a Parallel I/O File System:

1. Login to a client node as root.
2. Issue the **add_piofs_user** command (see “add_piofs_user” on page 40) for each user you wish to authorize.

Users not authorized in this manner cannot create new files in the file system, but, given sufficient standard AIX read and write permissions, they can read and write existing files.

To create directories for authorized users, use the AIX **mkdir** command.

Step 11: Set Environment Variable for When No License is Available

The Parallel I/O File System PIOFS_NOLIC_CMD environment variable optionally can contain a command to be issued when no license is available. To set the PIOFS_NOLIC_CMD environment variable in the Korn shell, issue:

```
export PIOFS_NOLIC_CMD=command_name
```

A C shell user can issue:

```
setenv PIOFS_NOLIC_CMD command_name
```

The environment variable can be set either by individual users or for all users in the /**etc**/**environment** file.

For more information about using the PIOFS_NOLIC_CMD environment variable, see “Handling License Availability” on page 33.

Step 12: Set Up Libraries for Compiling and Linking

On nodes where a Parallel I/O File System has been installed using **installp**, no action is necessary to set up the compiling and linking environment. If you (the system administrator) want to allow users to compile and link programs using the function calls provided with the Parallel I/O File System, then the following files must be made available as indicated.

Table 1. Setting Up the Compiling and Linking Environment for C and C++ Programs

Files installed at:	**Must be linked or copied to:**
/usr/lpp/piofs/include/piofs.h	/usr/include/piofs/piofs.h
/usr/lpp/piofs/include/piofs_ioctl.h	/usr/include/piofs/piofs_ioctl.h
/usr/lpp/piofs/include/piofs.exp	/usr/include/piofs/piofs.exp

Table 2. Setting Up the Compiling and Linking Environment for FORTRAN Programs

Files installed at:	**Must be linked or copied to:**
/usr/lpp/piofs/include/piofs.exp	/usr/include/piofs/piofs.exp
/usr/lpp/piofs/include/piofs.inc	/usr/include/piofs/piofs.inc
/usr/lpp/piofs/lib/libPiofsFortran.a	/usr/lib/piofs/libPiofsFortran.a

What To Do Next

If you are writing programs for the Parallel I/O File System, go to Chapter 7, “Nuts and Bolts of Using PIOFS” on page 61 to find out how to create and partition PIOFS files.

If you are managing a system, go to Chapter 4, “Administering the Parallel I/O File System” on page 29.

Managing the Parallel I/O File System

Chapter 4. Administering the Parallel I/O File System 29
Starting a Parallel I/O File System 29
Stopping a Parallel I/O File System 29
Mounting a Parallel I/O File System 29
Unmounting a Parallel I/O File System 30
Rebooting the System 30
Managing User Authorization and Access 30
Determining the Status of a Parallel I/O File System Server 31
Unconfiguring a Parallel I/O File System 31
Removing a Parallel I/O File System 32
Changing Your Configuration 32
Adding or Deleting a Disk or Node 33
Handling License Availability 33
Finding Information about Errors 34
Troubleshooting PIOFS Problems 34
Getting Help From IBM 36

Chapter 5. Restrictions, Limitations, and Deviations 37
Risk to Permanent or Mission-Critical Files 37
PIOFS Restrictions, Limitations, and Deviations 37
Connection Limitations 38
Limits 38

Chapter 6. Parallel I/O File System Commands 39

© Copyright IBM Corp. 1995

Chapter 4. Administering the Parallel I/O File System

This chapter describes the following tasks:

- Starting a Parallel I/O File System
- Stopping a Parallel I/O File System
- Mounting a Parallel I/O File System
- Unmounting a Parallel I/O File System
- Rebooting the system
- Managing user authorizing and access
- Determining the status of a Parallel I/O File System server
- Unconfiguring a Parallel I/O File System
- Removing a Parallel I/O File System
- Changing your configuration
- Adding or deleting a disk or node
- Handling license availability
- Troubleshooting PIOFS problems.

See Chapter 6, “Parallel I/O File System Commands” on page 39 for the complete syntax of the Parallel I/O File System commands.

Starting a Parallel I/O File System

Login to one of the servers as root and issue the command:

`/usr/lpp/piofs/bin/start_piofs_server` *`config_file_name`*

Use the *config_file_name* you created when you installed the Parallel I/O File System. You do not specify a path name for the configuration file. See “start_piofs_server” on page 52 for more information.

Stopping a Parallel I/O File System

Make sure you are logged in as root. From one of the servers, issue the command:

`/usr/lpp/piofs/bin/stop_piofs_server` *`config_file_name`*

You do not specify a path name for the configuration file. See “stop_piofs_server” on page 57 for more information.

Mounting a Parallel I/O File System

Mount a Parallel I/O File System on client nodes by issuing:

`mount /`*`your_mount_point`*

You must be root to issue this command. Mount the Parallel I/O File System on on every client node that needs to access the Parallel I/O File System. See “Step 9: Mount the Parallel I/O File System on All the Client Nodes” on page 23 for a full description of the the steps involved.

If you have more than one Parallel I/O File System, each one has its own mount point.

© Copyright IBM Corp. 1995

Unmounting a Parallel I/O File System

Use the **unmount** command to unmount the file system from a client node. If the file system is mounted on */your_mount_point* issue:

```
unmount /your_mount_point
```

You must be root to issue this command. If the file system is busy the command will fail.

Rebooting the System

Your servers must be started before mounting the Parallel I/O File System. You must mount the Parallel I/O File System whenever you reboot the system.

Managing User Authorization and Access

PIOFS provides commands that let you perform the following user-related tasks:

- Authorize a user to create PIOFS files
- Delete authorization for a user to create PIOFS files
- Determine the PIOFS resources (files, directories, cells) and limits associated with a user
- Change a user's block limit and object limit
- List the userid, block limit and object limit of all users of a particular Parallel I/O File System.

Authorizing a User to Create PIOFS Files

Users must be authorized to create files within a Parallel I/O File System (except for the root user who does not need specific authorization). To authorize a user to create PIOFS files, issue the **add_piofs_user** command (see “add_piofs_user” on page 40).

Users not authorized in this manner cannot create new files in the file system, but, given sufficient standard AIX read and write permissions, they can read and write existing files.

Deleting a User's Authorization to Create PIOFS Files

To delete a user's authorization to create PIOFS files, issue the **delete_piofs_user** command (see “delete_piofs_user” on page 42).

Determining the PIOFS Resources and Limits Associated with a User

To determine the PIOFS resources (files, directories, cells) and limits associated with a user, issue the **stat_piofs_user** command (see “stat_piofs_user” on page 55).

Changing a User's Block Limit and Object Limit

To change the block limit and object limit assigned to a user, issue the **change_piofs_user** command (see “change_piofs_user” on page 41).

Listing the Userid, Block Limit, and Object Limit for Each User of a PIOFS

To list the userid, block limit, and object limit of each user of a particular Parallel I/O File System, issue the **list_piofs_users** command (see “list_piofs_users” on page 46).

Determining the Status of a Parallel I/O File System Server

To determine the status of a mounted PIOFS server, issue the **stat_piofs_server** command (see “stat_piofs_server” on page 53).

The **stat_piofs_server** command returns the following information:

- The status of the server: available or not responding
- The logical server node number
- The hostname of the server node
- The number of logical disks
- The number of data blocks in use, the total number of allocated data blocks, and the percentage of data blocks used
- The percentage of used metadata space
- The number of files used
- The number of directories used
- The number of cells used.

This command is useful if the file system seems to be hung. The server status information synchronously gathers information about each node associated with the PIOFS server. If the command itself appears to be hung, it is probably timing out as it attempts to obtain status from a particular server node. When the time out completes, the hostname of the server node that failed to respond is shown.

See “Recovering from Server Node Failure or Server PIOFS Process Failure” on page 35 for additional information on how to handle server node failure.

Unconfiguring a Parallel I/O File System

This destroys all data files. If you need to save the data in files you must back them up or copy them to other file systems. You don't normally unconfigure a Parallel I/O File System.

To unconfigure a Parallel I/O File System:

1. Make sure no one is using the file system
2. Unmount the file system from each client node by issuing:

   ```
   unmount /your_mount_point
   ```

3. Stop the servers by issuing:

   ```
   stop_piofs_server config_file_name.
   ```

4. Issue:

   ```
   piofs_config -u  /var/piofs/config_file_name
   ```

Removing a Parallel I/O File System

Sometimes you wish to remove a product from your RS/6000 SP. The following are the actions you take to remove the Parallel I/O File System from your RS/6000 SP.

1. Unmount the file system from each client node by issuing:

   ```
   unmount /your_mount_point
   ```

2. Stop the servers by issuing:

   ```
   /usr/lpp/piofs/bin/stop_piofs_server config_file_name
   ```

3. Issue the following command from *each* client node to remove a Parallel I/O File System kernel extension:

   ```
   /usr/lpp/piofs/bin/installpiofs -d
   ```

4. Issue the following command from *each* client node and *each* server node to remove the Parallel I/O File System message passing kernel extension. (You need to issue this only once for a node acting as both client and server.)

   ```
   /usr/lpp/piofs/bin/installpmp -d
   ```

5. Unconfigure a Parallel I/O File System by issuing:

   ```
   piofs_config -u /var/piofs/config_file_name
   ```

6. Issue the **installp** command to remove the code from the system. Omit this step if you are removing the Parallel I/O File System to recover from a PIOFS failure.

   ```
   installp -r piofs (for AIX Version 3.2.5)
   installp -u piofs (for AIX Version 4.1)
   ```

Changing Your Configuration

Configuration creates logical volumes on each node on each disk specified in the configuration file. It also verifies the syntax of the configuration file and makes copies of the configuration files for the servers.

You can use the configure command for a new Parallel I/O File System, or to change an existing Parallel I/O File System. In either case the syntax is the same.

To change a configuration:

- Use **stop_piofs_server** to stop the file system
- Edit the configuration file
- Issue **piofs_config** again.

Caution

If you change the configuration, you will probably destroy any existing PIOFS files. Therefore, copy all existing PIOFS files to disk space outside the Parallel I/O File System before you reconfigure your system.

Adding or Deleting a Disk or Node

You need to reconfigure the file system when you add or delete a new disk or a new node. Do the following steps when you add or delete a disk or node:

1. Back up your files. This can be a difficult step if your files have been created with a file layout other than the default. For each file you need to:
   a. Make sure no one is writing to the file.
   b. Record the filename, basic striping unit (bsu), and the number of cells.
   c. Copy the file to another file system.
2. Unmount the file system from *each* client node:

   `unmount /your_mount_point`
3. Stop the servers:

   `/usr/lpp/piofs/bin/stop_piofs_server config_file_name`
4. Edit your configuration file, *config_file_name* to include the new or changed information.
5. Issue the configuration command, as shown below, to reconfigure the Parallel I/O File System. Run this command as root user from the same node where you created and edited the configuration file.

   `/usr/lpp/piofs/bin/piofs_config /var/piofs/config_file_name`

   Use the full path name for the configuration file when issuing the **piofs_config** command.
6. Format the servers by issuing:

   `/usr/lpp/piofs/bin/format_piofs_server config_file_name`
7. Start the servers with:

   `/usr/lpp/piofs/bin/start_piofs_server config_file_name.`

   See “Step 8: Start the Servers” on page 22 for more details about starting the reconfigured Parallel I/O File System.
8. Mount the file system:

   `mount /your_mount_point`
9. Restore your Parallel I/O File System files. You can use either of the following methods:
   - Issue `piofsioctl(..., PIOFS_CREATE,...);` to create a new file with the attributes saved in the first step.
   - Copy the saved files into the Parallel I/O File System.

Handling License Availability

You can access the Parallel I/O File System only when there is a license available for you to use. If there is no license available the server writes a message to **stderr**. The server also opens a pipe to the command defined in the **PIOFS_NOLIC_CMD** environment variable and writes the same message to that pipe. If **PIOFS_NOLIC_CMD** has not been defined, the command **mail root** is issued instead. In either case, the pipe is closed before the server exits.

A license may not be available if one of the following is true:

- The license server is not available.
- There are no licenses at all for the Parallel I/O File System.
- There is no nodelock license for the Parallel I/O File System.

Finding Information about Errors

PIOFS-related errors are logged to the following files, which are located in **/var/piofs**:

start_piofs_server.log This file is created when the **start_piofs_server** command loads the pmp kernel extension and finds that pmp has issued warnings that the specified network options differ from their recommended values. The nodes from which the warning messages were issued are listed. Use the start_piofs_server log to determine which nodes experienced errors. Then, login to each node listed and check its pmp.log to obtain more details on the errors.

The start_piofs_server log is created only on the node from which the **start_piofs_server** command is issued.

pmp.log This file is created on each node where the pmp kernel extension is loaded. If the node's network options differ from those recommended for PIOFS, warnings messages to that effect are logged in this file.

piofs_fsck.*config_file_name* This file is created on the server node from which **piofs_fsck** is issued. Messages issued by **piofs_fsck** are logged here.

Other errors are logged to the system error log. Use **errpt** or SMIT to view it.

Troubleshooting PIOFS Problems

Although we cannot provide troubleshooting and recovery procedures for all errors that you might encounter while using PIOFS, here are some tips on handling some of the errors.

Handling Configuration Problems on a Server Node

If you encounter a problem when you issue **piofs_config** to configure a Parallel I/O File System, it might be due to system logical storage resources (such as logical volumes) becoming out of sync with their ODM database entries. This is not actually a PIOFS problem but **piofs_config** will not work until it is corrected. Attempt to correct the problem by taking the following steps:

1. Unconfigure the Parallel I/O File System:

   ```
   piofs_config -u /var/piofs/config_file_name
   ```

2. Synchronize the logical volume control block, the device configuration database, and the volume group descriptor areas on the physical volumes:

```
synclvodm volume_group
```

3. Remove physical volumes from the volume group and delete the volume group if you have created a separate volume group for PIOFS:

   ```
   reducevg -df volume_group physical_volume
   ```

4. Omit this step until you have followed the rest of the procedure, re-attempted to configure the Parallel I/O File System and failed again. If this occurs, re-execute the entire procedure and include this step.

   Use **odm** or **odme** to find and remove PIOFS resources that might still be in the ODM database.

   The ODM subcomponents to check are :

   - **CuDv**
   - **CuAt**
   - **CuDep**

   The PIOFS entries to look for are:

   - The volume group defined for PIOFS to use
   - Logical volumes, **pfsd1** through **pfsd128**
   - Logical volume, **pfsmd**

5. Remove all file systems and/or special files in /**dev** that are associated with PIOFS resources using either of the following commands or equivalent ones:

   - **rmfs -r** *filesystem*,
     replacing *filesystem* with **pfsd1** through **pfsd128**, and **pfsmd**
   - **rmlv -f** *logical_volume*,
     replacing *logical_volume* with **pfsd1** through **pfsd128**, and **pfsmd**

6. Recreate the volume group:

   ```
   mkvg -f -y volume_group physical_volume
   ```

7. Vary on the volume group

   ```
   varyonvg volume_group
   ```

8. Configure the Parallel I/O File System

   ```
   piofs_config /var/piofs/config_file_name
   ```

Recovering from Client or Server Failure

The options and procedures for recovery depend on the cause of the failure and on whether the failure occurs on a client node or a server node.

Recovering from Client Node Failure

Try remounting the Parallel I/O File System.

Recovering from Server Node Failure or Server PIOFS Process Failure

When a server node fails or a server PIOFS process fails, pending PIOFS operations fail. In addition, any new operations that require the failing server node will also fail.

Issue the **stat_piofs_server** command to determine whether the node is still running. See “stat_piofs_server” on page 53 for information about the **stat_piofs_server** command.

If the problem node is still running and available, issue the following command on that node to determine if the server process is still running:

```
ps -ae | grep PiofsServer
```

If the server node is down or the PiofsServer process is not running, you will need to stop the server, correct any problems that might be causing the node to fail, issue **piofs_fsck** to repair PIOFS file system inconsistencies, and then restart the server.

Recovering from PIOFS Failure When Other Approaches Have Failed

If the Parallel I/O File System is in an unusable state and you cannot afford to reboot the server nodes, you might be able to recover by removing the Parallel I/O File System using the steps listed under “Removing a Parallel I/O File System” on page 32. If you follow this procedure, **do not perform the installp step**.

Getting Help From IBM

If you have difficulty installing a Parallel I/O File System or have any other questions related to this product, follow your local procedures for getting support.

If you are requested to call your IBM representative, please have the following information available to provide to the IBM representative:

- Your customer number
- The Parallel I/O File System product number (5765-297)
- The name and version of the operating system you are using
- A telephone number where you can be reached.

Chapter 5. Restrictions, Limitations, and Deviations

This section lists Parallel I/O File System restrictions and limitations, as well as PIOFS deviations from AIX. These are specific to the Parallel I/O File System and may not apply to other file systems at your installation.

Risk to Permanent or Mission-Critical Files

The Parallel I/O File System cannot be used to permanently store files. If any disk or any server node in a Parallel I/O File System fails, then the entire PIOFS fails. There is, in general, no recovery from disk failure except to recreate the files and restart the jobs that were running at the time of failure. For a Parallel I/O File Systems with *n* disks, the disk failure rate will be *n* times the failure rate of a single disk. Due to the potential for disk failure, the Parallel I/O File System should be used for temporary files or files that have been backed up on another file system.

PIOFS Restrictions, Limitations, and Deviations

Open flags

The only supported **open()** flags are:

O_RDONLY
O_WRONLY
O_RDWR
O_CREAT
O_EXCL
O_TRUNC

Any other open flag will be ignored. For instance, you cannot append to a Parallel I/O File System file.

Commands

The following commands are not supported for Parallel I/O File System files:

- **sync** and **fsync** — You cannot force modified PIOFS data to be saved to permanent storage. Modified PIOFS data is kept in multiple JFS files, which are inaccessible to users. JFS controls when the data is written to permanent storage.
- **mkfifo**
- **pathconf** and **pathfconf**

Memory mapping

Memory mapping of files is not supported. Because of this, source code placed in the Parallel I/O File System may not be compiled and linked depending on the binder you use. Also, executable files, placed in the Parallel I/O File System will not load and execute if they are larger than four Megabytes.

Record level locking

Record level locking is not supported. Specifically, **fcntl()** with the F_SETLK, F_SETLKW, or F_GETLK flags is not supported.

File types

Only regular files and directories are supported.

Hard links

Hard links are not supported.

© Copyright IBM Corp. 1995

Open directories A directory can be removed even if a user has the directory open.

File renaming A file cannot be renamed while it is open.

File size limits The Parallel I/O File System does not enforce the file size limitation set by **ulimit**.

Note

For a list of base AIX functions and subroutines that are changed or not supported by the Parallel I/O File System, see "Supported AIX Functions" on page 97.

Connection Limitations

Servers must use the same adapter types. For example, you cannot have some nodes switch-attached and some Ethernet-attached. For best performance, the client and server nodes should be switch-attached.

Limits

Each Parallel I/O File System has the following limits:

Table 3. Parallel I/O File System Limits

Maximum number of:	**Limit**
Server nodes	512
Logical volumes per server node	128
Gigabytes per server node	256
Bytes in a basic striping unit	2 Gigabytes (2,147,483,647 bytes)
Characters in an absolute path name	255
Cells per file per server node	255

Chapter 6. Parallel I/O File System Commands

add_piofs_user 40
change_piofs_user 41
delete_piofs_user 42
format_piofs_server 43
installpiofs 44
installpmp 45
list_piofs_users 46
piofs_config 47
piofs_fsck 49
piofs_verify 51
start_piofs_server 52
stat_piofs_server 53
stat_piofs_user 55
stop_piofs_server 57

© Copyright IBM Corp. 1995

add_piofs_user

Purpose

Authorize a user to create files in a Parallel I/O File System.

Syntax

add_piofs_user *name* *mount_point* [*block_limit* *object_limit*]

Parameters

name The name of the user to be authorized.

mount_point The mount point of the Parallel I/O File System to which the user is to be authorized.

block_limit The number of 64K blocks the user is to be allowed on a single server node.

If you omit this parameter, the default block limit, specified in the configuration file, is assigned to the user. You can also obtain the default value by coding a -1.

object_limit The number of objects that the user is to be allowed on a single server node.

If you omit this parameter, the default object limit, specified in the configuration file, is assigned to the user. You can also obtain the default value by coding a -1.

See “Sample Configuration File” on page 18 for information about the configuration file.

Description

The **add_piofs_user** command authorizes a user to create files in a Parallel I/O File System.

Return Codes

The **add_piofs_user** command returns one of the following:

0 Successful completion. The specified user was authorized to create files in the specified Parallel I/O File System.

-1 An error occurred.

Related Information

delete_piofs_user, **stat_piofs_user**, **change_piofs_user**, **list_piofs_users**

change_piofs_user

Purpose

Change a user's block limit and object limit.

Syntax

change_piofs_user *name mount_point block_limit object_limit*

Parameters

name The name of the user whose block limit and object limit are to be changed.

mount_point The mount point of the user's Parallel I/O File System

block_limit The number of 64K blocks the user is to be allowed on a single server node.

If you omit this parameter, the default block limit, specified in the configuration file, is assigned to the user. You can also obtain the default value by coding a -1.

object_limit The number of objects that the user is to be allowed on a single server node.

If you omit this parameter, the default object limit, specified in the configuration file, is assigned to the user. You can also obtain the default value by coding a -1.

See “Sample Configuration File” on page 18 for information about the configuration file.

Description

The **change_piofs_user** command changes a user's block limit and object limit.

Return Codes

The **change_piofs_user** command returns one of the following:

0 Successful completion. The specified user was given the new block limit and object limit.

-1 An error occurred.

Related Information

add_piofs_user, **delete_piofs_user**, **stat_piofs_user**, **list_piofs_users**

delete_piofs_user

Purpose

Delete a user's authorization to create files in a Parallel I/O File System.

Syntax

delete_piofs_user *name mount_point*

Parameters

name	The name of the user whose authorization is to be deleted.
mount_point	The mount point of the Parallel I/O File System to which the user is no longer to be authorized.

Description

The **delete_piofs_user** command deletes a user's authorization to create files in a Parallel I/O File System. Users whose authorization has been deleted can still access existing PIOFS files they own. However, they cannot create new PIOFS files or directories.

After you delete a user's authorization, you can no longer issue **stat_piofs_user** to obtain information about the PIOFS resources and limits associated with that user. The system removes this information when the user's authorization is deleted.

Return Codes

The **delete_piofs_user** command returns one of the following:

0 Successful completion. The specified user's authorization has been deleted.

-1 An error occurred.

Related Information

add_piofs_user, **stat_piofs_user**, **change_piofs_user**, **list_piofs_users**

format_piofs_server

Purpose

Format Parallel I/O File System servers.

Syntax

format_piofs_server [–**n**] *config_file_name*

Parameters

–**n**	Suppress the warning prompt. Use this flag with care!
config_file_name	The name of the configuration file.

Description

This command formats the Parallel I/O File System servers.

Caution

The **format_piofs_server** command destroys data on the formatted server. Use it only with extreme caution.

This command can be used for two purposes:

- To format PIOFS servers after you configure them.
- To re-format a formatted PIOFS server so it can be reused. Note that **format_piofs_server** erases all existing data on a formatted PIOFS file system and removes all user authorizations that were added using the **add_piofs_user** command.

Return Codes

The **format_piofs_server** command has the following return codes:

0 The server was formatted.
1 Reserved.
2 An error occurred invoking **format_piofs_server**.
3 An error occurred reading *config_file_name*.
4 Could not **ping** one of the server nodes.
5 Reserved.
6 An error notification was returned by **rsh**.
7 Error formatting the server.

Related Information

start_piofs_server, **stop_piofs_server**, **stat_piofs_server**

installpiofs

Purpose

Install or uninstall the Parallel I/O File System kernel extension.

Syntax

installpiofs [–**a** | –**d** | –**q**]

Parameters

–**a** Install the kernel extension if not already installed.

–**d** Remove the kernel extension.

–**q** Query to see if the kernel extension is installed.

Description

This command installs, removes, or queries the Parallel I/O File System kernel extension. This command is not part of the typical installation process; it is provided for service operations. The query returns a message indicating whether the kernel extension is installed.

Return Codes

The **installpiofs** command has the following return codes:

0 The operation was successful.
1 An error occurred invoking **installpiofs**.

installpmp

Purpose

Install or uninstall the Parallel I/O File System message passing kernel extension.

Syntax

installpmp { –**ai** *ipaddress* | –**d** | –**q** }
[–**p** *msg_port*
–**o** *ack_port*
–**r** *udp_recvspace*
–**s** *udp_sendspace*
–**t**
–**c** *config_file_name*]

Parameters

You must specify one of the first three parameters.

–**ai** *ipaddress*	Install the kernel extension if not already installed. The *ipaddress* is the Internet address in decimal, octal, or hex, of the server node.
–**d**	Uninstall the message passing kernel extension.
–**q**	Query the message passing kernel extension.

If you specify –**ai** *ipaddress*, you may optionally specify any of the following:

–**p** *msg_port*	The IP port number to use to pass messages.
–**o** *ack_port*	The IP port number to use to pass acknowledgements.
–**r** *udp_recvspace*	The UDP socket receive buffer size in bytes.
–**s** *udp_sendspace*	The UDP socket send buffer size in bytes.
–**t**	Indicates this is a server node.
–**c** *config_file_name*	The configuration file name.

Description

This function installs, removes, or queries the Parallel I/O File System kernel extension for Parallel I/O File System message passing.

This command is not part of the typical installation process; it is provided for service operations.

Return Codes

The query returns a message indicating whether the kernel extension is installed.

The **installpmp** command has the following return codes:

0 The operation was successful.
1 An error occurred invoking **installpmp**.

list_piofs_users

Purpose

List the userid, block limit, and object limit of all users of a particular Parallel I/O File System.

Syntax

list_piofs_users *mount_point*

Parameters

mount_point The mount point of the Parallel I/O File System whose users you wish to list.

Description

The **list_piofs_users** command lists the userid, block limit, and object limit of all users of a particular Parallel I/O File System.

uid The user's userid

user name The user's user name

block_limit The user's block limit per server

object_limit The user's object limit per server.

This information is returned as message INFO211. See Appendix A, “Parallel I/O File System Messages” on page 213 for a description of the message.

Return Codes

The **list_piofs_users** command returns one of the following:

0 Successful completion.

-1 An error occurred.

Related Information

add_piofs_user, **stat_piofs_user**, **change_piofs_user**, **delete_piofs_user**

piofs_config

Purpose

Configure server nodes using a *config_file_name*.

Syntax

piofs_config [–**u**] [–**n**] [–**f**] *config_file_name*

Parameters

–**u**	Unconfigure the server nodes. This is a destructive change.
–**n**	Suppress the warning prompt.
–**f**	Update only the active *config_file_name* on this node and in the system data repository. Don't physically configure any disks.
config_file_name	A configuration file created in “Step 4: Create the Configuration File” on page 17. If you have configured multiple Parallel I/O File Systems, use the configuration file name for the file system you wish to configure.

Description

The **piofs_config** script configures server nodes for PIOFS by creating files on each of the disks on each of the server nodes specified in the configuration file. These files are used by PIOFS; they are not user files. Users' files will be created by PIOFS as data in the files created by the **piofs_config** script.

When you omit the **-u** flag, the **piofs_config** command builds the necessary files at each node as specified by the configuration file. **piofs_config** checks each node to see if the configuration file specifies anything different for each node from what is already configured at that node. If there are no differences, no work is done.

When you specify the –**u** flag, the **piofs_config** command removes from the server nodes *all* the files created during configuration. If only some of the server nodes are to be removed use an appropriate new configuration file.

Note: No files are removed from client nodes. If you change or unconfigure and are not using the SDR, you should also change or remove the configuration files at the client nodes.

See “Sample Configuration File” on page 18 for the syntax of a configuration file.

Return Codes

The **piofs_config** command returns one of the following:

0 Successful completion. The verified and active configuration file is /**etc**/*config_file_name***.active**. The active IPaddress file is saved as /**etc**/*config_file_name* **.IPaddress.active**.

1 Invalid invocation.

2 Error in configuration file.

3 One or more nodes failed to configure.

Examples

1. To configure everything in the /**var**/**piofs**/*config_file_name*:

   ```
   /usr/lpp/piofs/bin/piofs_config /var/piofs/config_file_name
   ```

2. To change the configuration (assuming the two files specify different configurations):

   ```
   /usr/lpp/piofs/bin/piofs_config  /var/piofs/another_config_file
   ```

3. To unconfigure all the storage nodes:

   ```
   /usr/lpp/piofs/bin/piofs_config -u /var/piofs/config_file_name
   ```

piofs_fsck

Purpose

Check PIOFS file system consistency and, if possible, correct errors.

Syntax

piofs_fsck [–n] *config_file_name*

Parameters

–n	Report errors found but don't attempt to correct them.
config_file_name	The name of the configuration file associated with the PIOFS file system to be checked.

Description

The **piofs_fsck** command attempts to verify that:

- All objects listed in directories exist
- All file, directory, and symbolic link objects are listed in a directory (except for "/")
- All directory names are prefixes of the names of their contained objects
- No object is listed in more than one directory
- No two files, directories, or symbolic link objects have the same ID
- No two files, directories, or symbolic link objects have the same path name
- All cells belong to a file
- All cells of each file exist.

If file system inconsistencies are found and you have specified the –n option, **piofs_fsck** reports the errors but does not attempt to repair them.

If file system inconsistencies are found and you have omitted the –n option, **piofs_fsck** reports the errors and attempts to repair them. Orphaned files and directories (those that cannot be reached) are reconnected by placing them in the directory defined by their full path name.

Errors that require user action are issued to standard output and logged to /**var**/**piofs**/**piofs_fsck.log.***config_file_name*. Errors that are only intended for IBM service are logged but not displayed or documented.

Note

The **piofs_fsck** command must be issued from a server node. Before issuing the **piofs_fsck** command, the PIOFS file system must have been stopped using the **stop_piofs_server** command.

If you request that corrective actions be taken, be aware that some loss of data might occur. **piofs_fsck** is conservative in its repair efforts and tries to avoid

actions that might result in the loss of valid data. Always run **piofs_fsck** after a system malfunction

Return Codes

The **piofs_fsck** command has the following return codes:

0 No inconsistencies were found in the file system.
4 Inconsistencies were found but they were corrected.
8 Inconsistencies were found but some could not be corrected.
12 **piofs_fsck** could not run successfully.

piofs_verify

Purpose

Verify the Parallel I/O File System installation.

Syntax

piofs_verify [–**q** | –**v**]

Parameters

–**q** Write output to the file /**tmp**/**piofs_verifylog**. Do not write output to stdout.

–**v** Write output to stdout.

Description

This command verifies the Parallel I/O File System installation. It creates a Parallel I/O File System file, then deletes the file.

You must be **root** to run this command.

Return Codes

The **piofs_verify** command has the following return codes:

0 The command completed successfully.
1 The command was not issued by **root**.
2 A system resource or program was not found.
3 An error occurred and the verification could not continue.
4 An error occurred in the cleanup process.

Files

/**tmp**/**piofs_verifylog** Contains the output when –**q** is specified.

start_piofs_server

Purpose

Start the Parallel I/O File System servers.

Syntax

start_piofs_server *config_file_name*

Parameters

config_file_name The name of the configuration file.

Description

This command starts the Parallel I/O File System servers. The file system must have already been formatted.

Return Codes

The **start_piofs_server** command has the following return codes:

0 The servers started successfully.
1 The servers had already been started.
2 An error occurred invoking **start_piofs_server**.
3 An error occurred reading *config_file_name*.
4 Could not ping one of the server nodes.
5 An error occurred installing the **pmp** kernel extension.
6 An error notification was returned by **rsh**.
7 An error occurred starting the servers.

Related Information

format_piofs_server, **stop_piofs_server**, **stat_piofs_server**

stat_piofs_server

Purpose

Determine the status of a mounted PIOFS server.

Syntax

stat_piofs_server [–**l**] *mount_point*

Parameters

–**l**	Display the long form of the server status information.
mount_point	The mount point of the Parallel I/O File System.

Description

The **stat_piofs_server** command returns the following information about each node associated with the PIOFS server:

status	The status of the server node: **AVAIL** The server is available. **NORESP** The server is not responding.
server	The logical server node number.
host	The hostname of the server node.
disks	The number of logical disks.

If you requested the long form of the server status information, the following additional information is also returned:

blocks	The number of data blocks in use, the total number of allocated data blocks, and the percentage of data blocks used.
metadata	The percentage of used metadata space.
files	The number of files used.
dirs	The number of directories used.
cells	The number of cells used.

The short form of the server status information is returned as message INFO214. If you requested the long form of the server status information, INFO214 is displayed followed by additional information in message INFO215. See Appendix A, “Parallel I/O File System Messages” on page 213 for a description of the messages.

This command is useful if the file system seems to be hung. The server status information synchronously gathers information about each node associated with the PIOFS server. If the command itself appears to be hung, it is probably timing out as it attempts to obtain status from a particular server node. When the time out completes, the hostname of the server node that failed to respond is shown.

See “Recovering from Server Node Failure or Server PIOFS Process Failure” on page 35 for additional information on how to handle server node failure.

Return Codes

The **stat_piofs_server** command returns one of the following:

0 Successful completion.

-1 An error occurred.

Related Information

start_piofs_server, **format_piofs_server**, **stop_piofs_server**

stat_piofs_user

Purpose

Determine the PIOFS resources and limits associated with a user.

Syntax

stat_piofs_user *name mount_point*

Parameters

name The name of the user whose resources and limits are to be returned.

mount_point The mount point of the Parallel I/O File System that the user is using.

Description

The **stat_piofs_user** command returns the following information about the PIOFS resources and limits associated with a user:

uid The user's userid

block_limit The user's block limit (includes the space on all servers).

blocks_used The number of blocks used.

object_limit The user's object limit (includes the total number of objects on all the servers: files, directories, and cells).

files The number of files used.

dirs The number of directories used.

cells The number of cells used.

This information is returned as message INFO208 (AIX Version 4.1) or message INFO209 (AIX Version 3.2.5). See Appendix A, “Parallel I/O File System Messages” on page 213 for descriptions of the messages.

Note that after you delete a user's authorization (using **delete_piofs_user**), you can no longer issue **stat_piofs_user** to obtain information about the PIOFS resources and limits associated with that user. The system removes this information when the user's authorization is deleted.

Return Codes

The **stat_piofs_user** command returns one of the following:

0 Successful completion.

-1 An error occurred.

stat_piofs_user

Related Information

add_piofs_user, **delete_piofs_user**, **change_piofs_user**, **list_piofs_users**

stop_piofs_server

Purpose

Stop the Parallel I/O File System servers.

Syntax

stop_piofs_server *config_file_name*

Parameters

config_file_name The name of a configuration file.

Description

This command stops the Parallel I/O File System servers.

Return Codes

The **stop_piofs_server** command has the following return codes:

0 The servers have been stopped.
1 The servers had already been stopped.
2 An error occurred invoking stop_piofs_server.
3 An error occurred reading *config_file_name*
4 Reserved.
5 Reserved.
6 An error notification was returned by **rsh**.
7 An error occurred stopping the servers.

Related Information

start_piofs_server, **format_piofs_server**, **stat_piofs_server**

stop_piofs_server

Using the Parallel I/O File System

Chapter 7. Nuts and Bolts of Using PIOFS 61
Why Use Dynamic File Partitioning? 61
Overview of Dynamic File Partitioning 62
Useful Applications for a Parallel I/O File System 64
Understanding File Partitioning 65
Using PIOFS and AIX Functions to Create and Partition Files 71
Simple, Effective Ways to Partition a PIOFS File 72

Chapter 8. Programming Considerations 85
Understanding Atomicity of Reads and Writes 85
Controlling Concurrent Access to Parallel I/O File System Files 85
Maximizing PIOFS I/O Performance 88
Checkpointing Files 89
Specifying File Attributes for Parallel I/O File System Files 89
Accessing PIOFS Files Correctly 92
Using Existing Programs with a PIOFS File 94

Chapter 9. Using PIOFS Functions and Base AIX Functions 95
Using PIOFS Functions 95
Supported AIX Functions 97

© Copyright IBM Corp. 1995

Chapter 7. Nuts and Bolts of Using PIOFS

This chapter answers two questions: how can your applications get the most advantage from using PIOFS and what are the steps involved in creating and using PIOFS files?

In many cases, you will begin by using your PIOFS files as normal AIX files. With this approach, you can exploit immediately several key PIOFS features. You will be able to:

- Create files greater than 2 GB
- Achieve moderate parallelism, and hence better I/O performance, by spreading files across multiple server nodes
- Capture and save snapshots of PIOFS files during processing. If your program fails, you can use the checkpointed version of the file to restart the program where it was last checkpointed. You can also use file checkpointing to save a particular version of a PIOFS file.

Why Use Dynamic File Partitioning?

Many serial programs, written to run on a uniprocessor, can be easily parallelized just by running a copy of the program simultaneously on several different nodes. Small changes to the program are usually needed, for instance, to allow global data to be shared. However, if the program performs file I/O, the conversion to a parallel program is often more difficult.

The problem is that the multiple copies of the program need simultaneous access to the same file. As a result, I/O bottlenecks significantly diminish the value of running the application in parallel.

One solution to this problem is to have each copy of the program use a separate file. However, this technique introduces a different problem: administrative complexity. All those separate files on separate systems must be managed and maintained.

By taking the next step and using dynamic file partitioning, you can fully parallelize your application, both physically and logically. You can also convert an application from serial to parallel with only minimal re-coding. PIOFS functions are provided to let you implement complex algorithms with less complicated programming required.

Dynamic file partitioning lets you divide a PIOFS file into multiple, disjoint subfiles and have each subfile processed simultaneously by a different process in your application. Each subfile appears as a normal AIX file to its associated process.

Before we get to the details of creating PIOFS files and partitioning them, let's look at some examples of computational problems that can be performed better using PIOFS dynamic file partitioning. These examples will help you understand how PIOFS can be used.

After we've presented the examples and you have a better idea of how you can use PIOFS, we'll go into the details of how to create and partition PIOFS files. After that, we'll present sample programs showing how to code the file partitioning

© Copyright IBM Corp. 1995

functions. Along the way, we'll also discuss what you need to consider when you are deciding the best way to partition files for your application.

Overview of Dynamic File Partitioning

To follow the examples, you'll need to understand some basic concepts related to dynamic partitioning. Let's look at a PIOFS file containing some numbers, such as the one shown in Figure 2:

```
 1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
 .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .
```

Figure 2. A File of Numbers

Figure 2 shows the logical view of a sample PIOFS file. Conceptually, a PIOFS file is a two-dimensional matrix of data units that you can logically apportion into multiple subfiles, each to be accessed in parallel by different processes of a parallel application.

The horizontal dimension of the file is fixed at the time you create the file. The vertical dimension of the file is extensible and is limited only by the amount of disk space available in the Parallel I/O File System. In reality, parts of the file could be stored on several different server nodes, but with PIOFS, you don't have to worry about that.

How do you specify which units of data go in each subfile? PIOFS dynamic file partitioning lets you logically divide a file into subfiles in many different ways called *views*.

Each view consists of a different arrangement of the data into subfiles. For instance, here are a few possible views of the file in Figure 2.

- A different subfile for each column (Figure 3 on page 63)
- A different subfile for each row (Figure 4 on page 63)
- Three different subfiles, with data assigned to each in round-robin fashion (Figure 5 on page 63):
  - Subfile 0 would be assigned 1, 4, 7, 10, ...
  - Subfile 1 would be assigned 2, 5, 8, 11 ...
  - Subfile 2 would be assigned 3, 6, 9, 12...
- Three subfiles, with three-number by two-number blocks of data assigned to each in round-robin fashion (Figure 6 on page 63):
  - Subfile 0 would be assigned 1, 2, 3, 10, 11, 12, 19 ..
  - Subfile 1 would be assigned 4, 5, 6, 13, 14, 15, 22 ...
  - Subfile 2 would be assigned 7, 8, 9, 16, 17, 18, 25 ...

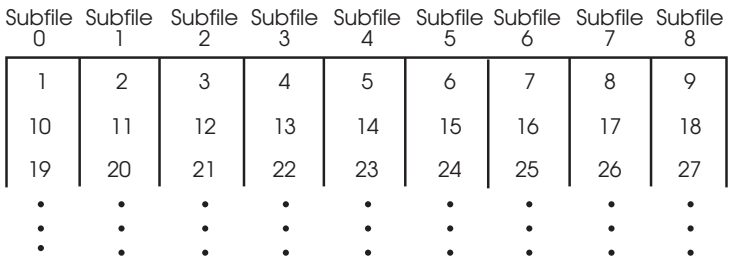

Figure 3. A PIOFS file partitioned by column

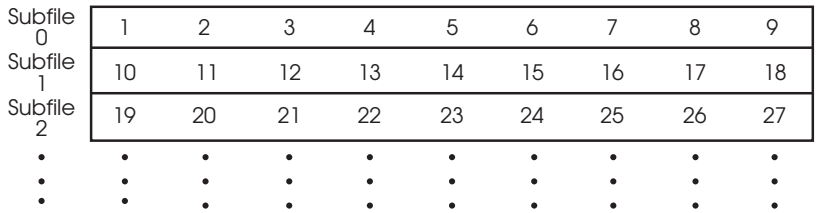

Figure 4. A PIOFS file partitioned by row

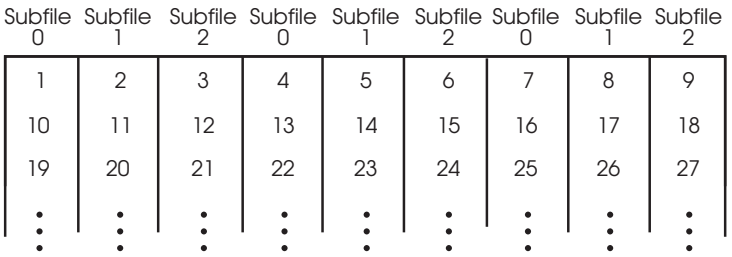

Figure 5. A PIOFS file partitioned by column using three subfiles

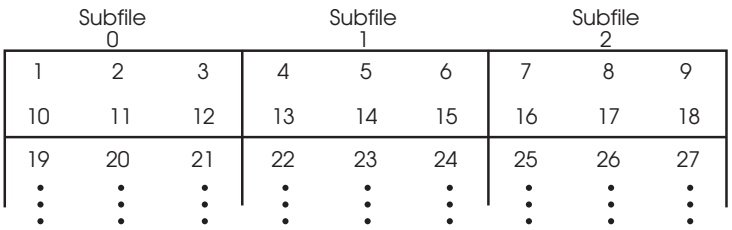

Figure 6. A PIOFS file partitioned using a 3 x 2 block pattern

You can partition the file in more complex ways, which will be covered later. For now, this is all you need to know.

Not only can a file have many different views, it can have more than one view at a time. That is, several processes can each have a different view of the same file. The way you divide the data into subfiles depends on your application.

Useful Applications for a Parallel I/O File System

The following short examples illustrate the type of computational problems that can be solved faster and more easily using PIOFS.

Sorting a Large Quantity of Data

One common task handled by applications is to sort a large quantity of data. You can use PIOFS to sort data quickly and efficiently by having a number of tasks perform the job in parallel.

Suppose you want to sort a file of integers using a Shear sort. You could treat the file of integers as a two dimensional grid. You could partition the file into two different views, once by rows of the grid and once by columns. In each view, you would have a number of subfiles equal to the number of processes in the program.

You would perform the sort as follows:

- Repeat the following actions for a number of interations equal to the logarithm base 2 of the number of rows of the file:
  - Sort the rows of the file (even-numbered rows in ascending order, odd-numbered rows in descending order).
  - Sort the columns of the file in ascending order.
- Sort all the rows in ascending order for the final pass to produce the sorted file.

Figure 7 illustrates the procedure.

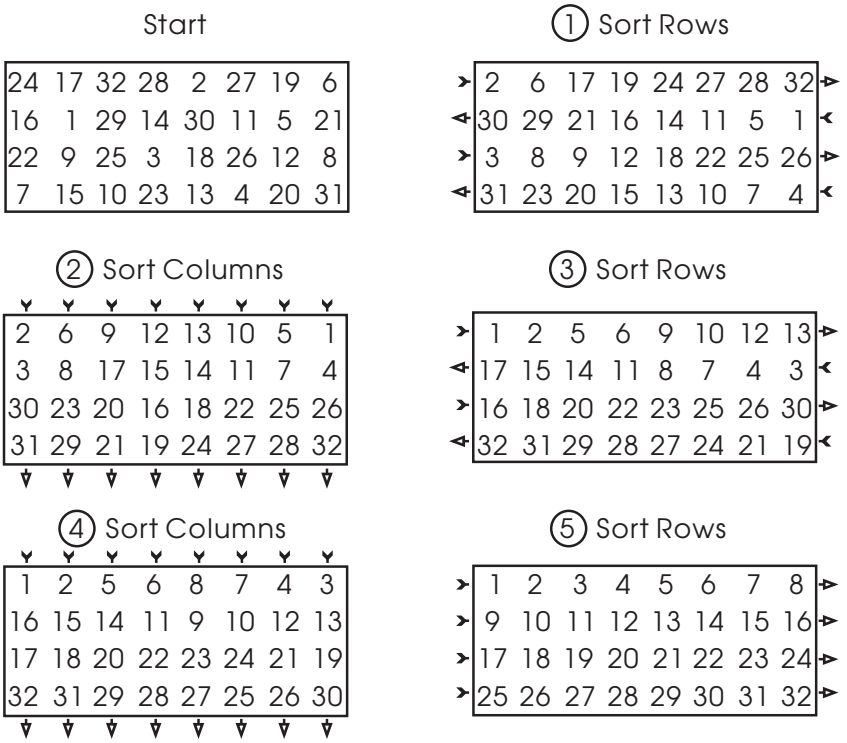

Figure 7. Steps involved in performing a Shear sort using a PIOFS file.

To see a coded example of a Shear sort using PIOFS dynamic file partitioning, see "Example of Sorting Data using PIOFS" on page 76.

Performing a Matrix Multiplication

A PIOFS file makes it easy for you to perform a matrix multiplication in parallel. Suppose you want to compute the product, C, of two square matrices A and B.

Your application could consist of four compute tasks running in parallel. Matrix A would be partitioned into two subfiles, each containing half the rows of the matrix. Matrix B would be partitioned into two subfiles, each containing half the columns of the matrix. Matrix C would be partitioned into four subfiles, each containing a quadrant of the matrix.

Each task would access:

- One subfile of matrix A
- One subfile of matrix B
- The subfile of matrix C that will contain the quadrant resulting from the computation of the product of the portion of matrix A and the portion of matrix B accessed by that task.

If N is the dimension of the matrices and the tasks are numbered 0, 1, 2 and 3, you could partition the matrices as shown in Figure 8.

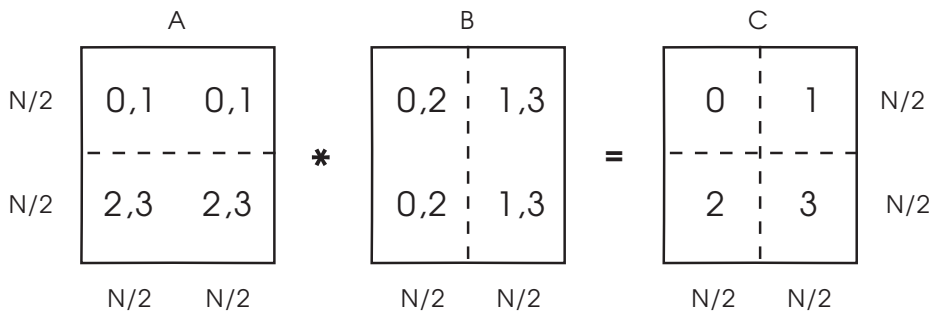

Figure 8. A matrix multiplication using partitioned PIOFS files. The operation uses four compute tasks (0, 1, 2, and 3).

To see a coded example of a matrix multiplication using PIOFS file partitioning, see "Example of Performing a Matrix Multiplication with PIOFS" on page 80.

Understanding File Partitioning

Now that you are more familiar with some of the ways a Parallel I/O File System can be used, let's go into the specifics of how you partition a file.

Assigning File Cells

A Parallel I/O File System file can contain data spread across multiple server nodes. The file's data is logically organized as a collection of cells, where a *cell* is a piece of the file stored on a particular server node. Multiple cells can reside on a particular server node but a cell cannot span multiple server nodes. Cells logically connect the different pieces of the file's data. If you think of the internal organization of a PIOFS file as a two-dimensional matrix, each cell comprises one column of the matrix.

When you create a Parallel I/O File System file, you specify the number of cells that will make up the file. The default is one cell per server node, which maximizes

an application's ability to access each piece of the file in parallel. The number of cells determine the maximum degree of parallelism the file can support. Once the file has been created, you cannot change the number of cells the file has.

If you specify more cells than there are nodes containing your file's data, the cells are assigned to the server nodes in a round-robin manner. Figure 9 shows a file consisting of four cells spread across two server nodes. Cells are numbered starting with 0. Because the cells were assigned in a round-robin manner, server node A contains both cell 0 and cell 2. To simplify the figure, the data in the file is represented as d's.

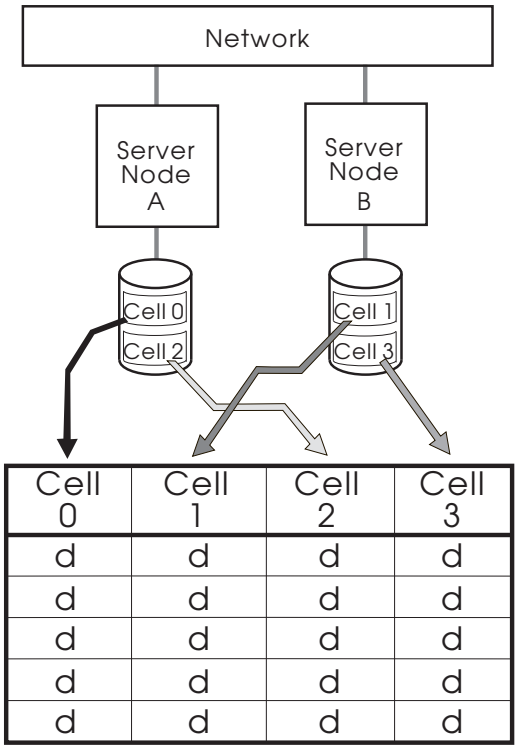

Figure 9. A simple PIOFS file: 4 cells spread across 2 server nodes.

Partitioning a File into Subfiles

Conceptually, you partition a PIOFS file into subfiles in much the same way as you would stamp out cookies from a batch of cookie dough and divide the cookies among a number of different cookie sheets.

Imagine the rectangle of rolled-out dough is the two-dimensional array of file data. Although in real life, you would probably measure the area of the dough in inches or centimeters, in terms of PIOFS file partitioning, the vertical dimension's unit of measure (that is, the smallest unit) is a *basic striping unit* or BSU. You specify the BSU of a PIOFS file when you create the file. The BSU can be up to 2 GB. The horizontal dimension's unit of measure is the cell. Figure 10 on page 67 shows the layout. To simplify the figure, the data is represented as d's.

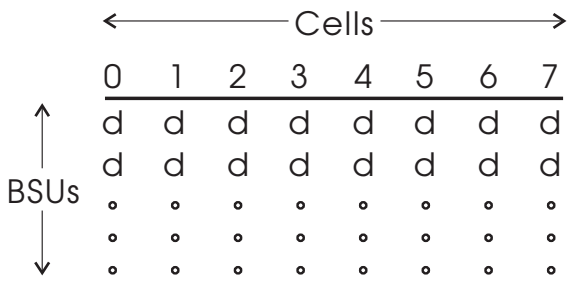

Figure 10. A logical representation of a PIOFS File.

You can stamp out cookies in different sizes. However, all the cookies for a particular batch of dough must be the same size. Cookies must be rectangular but you can adjust the horizontal and vertical dimensions of the cookie cutter to change the size of the cookie.

In terms of PIOFS file partitioning, a cookie is a *block*. The dimensions of the block are specified using two parameters:

- *Vertical block size* (Vbs), which is the height of the block in number of BSUs.
- *Horizontal block size* (Hbs), which is the width of the block in number of cells.

Each block is a unit of data to be assigned to one of the subfiles. You can think of the cookie sheets as the subfiles. You must stamp cookies out from left to right and each cookie goes on the next cookie sheet in round-robin fashion. You need not stamp out the cookies from the left margin of the dough all the way to the right. You can stamp out the cookies in different rectangular patterns. However, you must use the same pattern for the entire file.

In terms of PIOFS file partitioning, the data blocks are divided among the subfiles in a rectangular, repeating *pattern* as shown in Figure 11.

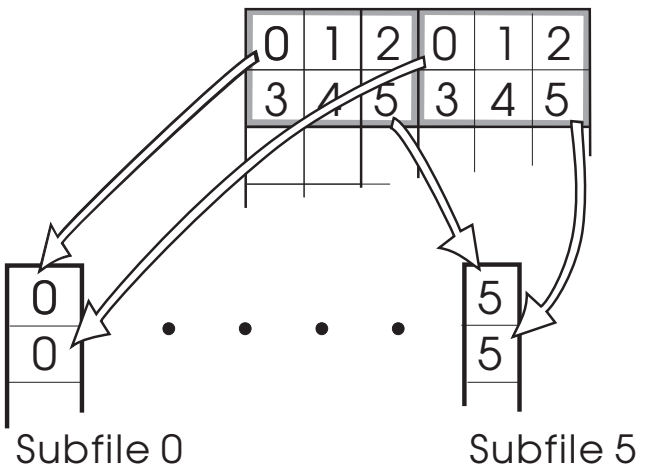

Figure 11. A PIOFS file being divided into subfiles. The numbers represent the subfile to which the block is assigned.

The shape and size of the pattern is specified using two parameters:

- *Vertical interleave* (Vn), which is the number of the blocks in the vertical dimension of the pattern. Another way of thinking about Vn: it specifies the number of subfiles interleaved in the same cell.

- *Horizontal interleave* (Hn), which is the number of blocks in the horizontal dimension of the pattern. Another way of thinking about Hn: it specifies the number of subfiles interleaved in the horizontal dimension.

One iteration of the pattern assigns a single block to each subfile. In Figure 11 on page 67 the pattern is specified by Hn=3, Vn=2, or a 3 x 2 block matrix.

Now, let's summarize what we've covered so far.

The Four PIOFS File Partitioning Parameters

The following parameters determine how a PIOFS file is partitioned. These are:

Vbs The height of the block in number of BSUs

Hbs The width of the block in number of cells

Vn The height of the block pattern in blocks

Hn The width of the block pattern in blocks.

Note: The number of subfiles is equal to Hn x Vn.

Figure 12 on page 69 shows how all the PIOFS file partitioning parameters we've discussed fit together.

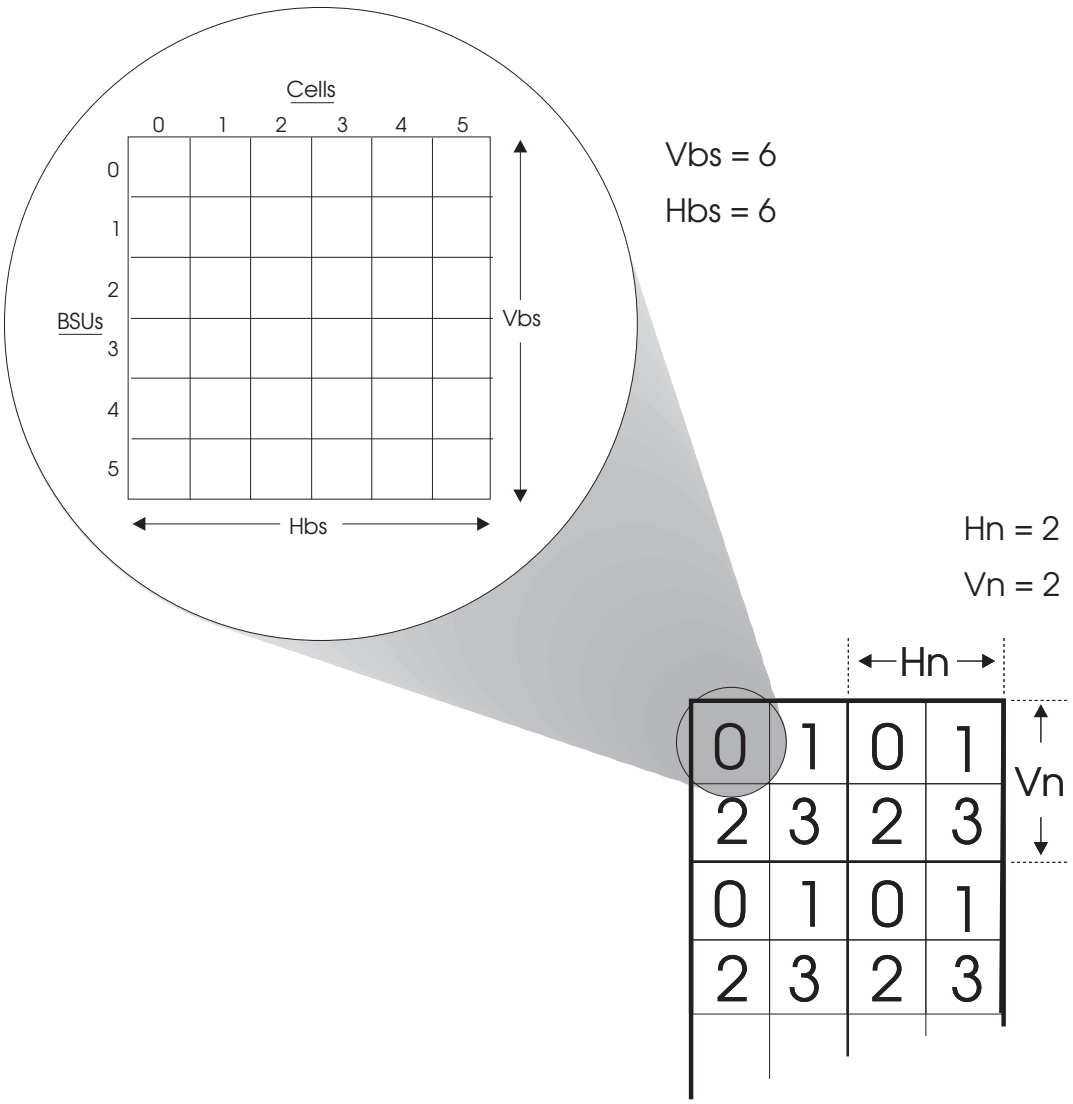

Figure 12. PIOFS file partitioning parameters and how they fit together. This PIOFS file consists of 24 cells. As shown in the exploded view of the first block, each block consists of six cells.

Rules Relating to File Partitioning

File partitioning follows some basic rules:

- Cells are numbered starting with zero.
- Within each subfile:
  - BSUs in a block are numbered (and read and written) in *column-major* order as shown in Figure 13.
  - Blocks are numbered (and read and written) in *row-major* order as shown in Figure 14.
- Blocks are assigned to subfiles in row-major order.
- Subfiles are numbered from 0 to (Hn x Vn) - 1 in row-major order in an Hn x Vn rectangle of blocks.

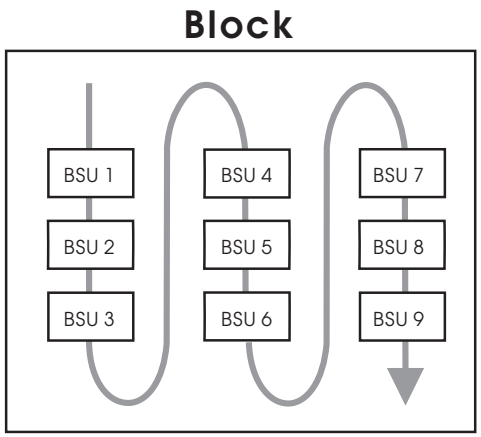

Figure 13. Logical order of BSUs in a block. BSUs are accessed in column-major order.

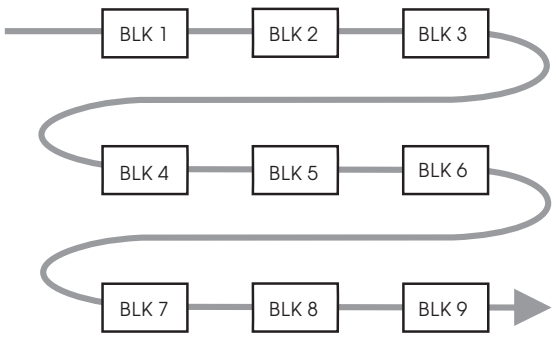

Figure 14. Logical order of blocks in a subfile. Blocks are accessed in row-major order.

Handling Empty or Incompletely Filled Blocks

Each subfile in a partitioned PIOFS file must map to at least one cell. To ensure that a PIOFS file meets this requirement, the parameter values for Hbs and Hn must satisfy the following condition:

```
(Hbs x (Hn - 1)) + 1  ≤ # of cells
```

If the number of cells is not evenly divisible by (Hn x Hbs), PIOFS logically pads the file with extra cells called *ghost cells*. For instance, suppose you partition a file using:

- 11 cells
- A block size of Hbs=3 and Vbs=2
- A block pattern of Hn=4 and Vn=1
- Four (Hn x Vn) subfiles

Hbs=3 and Hn=4 specify that each consecutive group of three cells be assigned to the next of four subfiles. Because Hn x Hbs = 12 and there are only 11 cells, the number of cells is **not** evenly divisible by Hn x Hbs. As a result, the file requires a single ghost cell (cell 11) as shown in Figure 15.

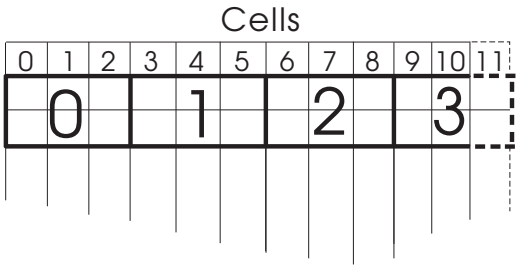

Figure 15. A PIOFS File divided into 4 subfiles with a ghost cell (cell 11).

Ghost cells do not really exist. PIOFS uses them to maintain consistent offsets for data within each subfile. That way, parallel programs that need to treat a PIOFS file as a single, two-dimensional matrix can safely predict the offset at which each block associated with a particular subfile is located.

Even though ghost cells contain no actual data, their "contents" take up space in your buffer. However, ghost cells cannot be read from or written to. The count of bytes returned from a read or write does not include the bytes corresponding to the ghost cells. However, when you allocate buffers to read or write data, you must reserve space for the ghost cells.

For instance, to read from or write to the block in Figure 15 that contains the ghost cell, you need the same sized buffer as you would for the other blocks. However, the count returned from a read or write will reflect only the bytes occupied by the real cells.

Using PIOFS and AIX Functions to Create and Partition Files

Now that you understand what file partitioning is, let's discuss the PIOFS and AIX functions you'll be using to create and partition the files. These functions are all covered in greater detail in Chapter 8, “Programming Considerations” on page 85 and in the programming reference section of this book. However, you'll need some general information on these functions to understand the coding examples that follow.

Here, we refer to PIOFS functions using their C language names and parameters but you can map these easily to their FORTRAN equivalents, which are all described immediately after each PIOFS C function call in the programming reference section of the book.

You create a PIOFS file and specify the number of cells it is to have using **piofsioctl** with the PIOFS_CREATE option. Once you have created the file, you

have the option of partitioning it into subfiles using **piofsioctl** with the PIOFS_CHANGE_VIEW option.

Alternatively, you can create the file using the AIX **open()** call. In this case, the PIOFS file is created with the attributes specified in the **piofs.profile** (discussed in "Specifying File Attributes for Parallel I/O File System Files" on page 89). If there is no **piofs.profile**, the file is assigned the default attributes. You would probably want to create the file using **open()** if you were running an existing program against a PIOFS file because it would require less modification of the program.

Simple, Effective Ways to Partition a PIOFS File

There are two very simple ways to partition a PIOFS file. One way is to partition the file into subfiles where each subfile corresponds to a single cell in the PIOFS file. This is called *column partitioning*. The second way is to partition the PIOFS file into subfiles across cells. This is called *row partitioning*.

Column Partitioning

In column partitioning, as shown in Figure 16, the cells of the file form the columns of the two-dimensional matrix comprising the logical view of the file.

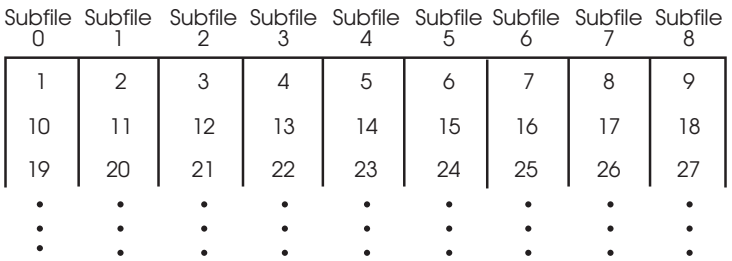

Figure 16. A PIOFS file partitioned by column - one cell per subfile

To achieve the most parallelism, partition the PIOFS file so that:

- Each cell is located on a different server node
- Each subfile corresponds to a single cell
- Each subfile is processed by a separate process.

This structure ensures that I/O accesses to each subfile involve different servers, avoiding delays caused by server contention. Also, if each subfile is opened by a different process and the I/O load of each process is uniform, the load is balanced across the server nodes so you can achieve good performance.

The example below illustrates the use of column partitioning. It shows a program creating a PIOFS file, partitioning it into subfiles by column, and processing each subfile in parallel.

Example of Column Partitioning

The example program consists of multiple processes running simultaneously on a number of nodes. The processes work in parallel to fill a PIOFS file with random numbers. Each process writes to a different cell mapped to its own subfile. Subfiles are filled with random numbers using the following method, which is repeated 1000 times:

1. A 500-element block (array) is filled with random numbers. This is the BSU for the file.
2. The contents of the array (the BSU) are written to the subfile.

The PIOFS file is partitioned as follows:

```
# of cells = # of processes = # of subfiles
```

The program below uses subroutines that are part of the IBM AIX Parallel Environment program product but you can substitute equivalent functions.

```
/* Simple program to fill a large file with random integers. */
#include <piofs/piofs_ioctl.h>
#include <errno.h>
#include <stdlib.h>
#include <mpproto.h>
#define BLOCK_SIZE 500
#define NUMBER_OF_BLOCKS 1000
main () {
   int my_proc_num, number_of_procs, fd, i, j;
   int block[BLOCK_SIZE];
   int allgrp, nbuf[4];
   piofs_create_t piofs_create;
   piofs_change_view_t piofs_change_view;

   /* Determine the total number of processes and the number of this
      process. my_proc_num is a number in the range 0 to
      number_of_procs - 1, which we will later use to specify the
      subfile to open. */
   if ( mpc_environ( &number_of_procs, &my_proc_num ) != 0 )
      exit( mperrno );

   /* Obtain the group ID of the group that includes all processes. */
   if ( mpc_task_query( nbuf, 4, 3 ) != 0 )
      exit( -1 );
   allgrp = nbuf[ 3 ];

   /* Process 0 creates the PIOFS file to hold the random numbers. */
   if ( my_proc_num == 0 ) {
      /* Create the PIOFS output file. */
      strcpy (piofs_create.name,"random_int_output_file");
      piofs_create.bsu = BLOCK_SIZE * sizeof( int );
      piofs_create.cells = number_of_procs;
      piofs_create.permissions = 0644;
      piofs_create.base_node = -1;
      piofs_create.flags = 0;

      if ( piofsioctl( 0, PIOFS_CREATE, &piofs_create ) != 0 )
         exit( errno );
   }

   /* Synchronize all processes before opening the file to ensure that
      the file has been successfully created before we issue any open
      requests. */
   if ( mpc_sync( allgrp ) )
      exit( -1 );
```

```
   /* Open the file to get a file descriptor. */
   fd = open( "random_int_output_file", O_WRONLY, 0 );
   if ( fd == -1 )
      exit( errno );

   /* Change the view of the file so that each process has
      its own subfile. Each subfile is a cell. */
   piofs_change_view.Vbs = 1;
   piofs_change_view.Vn  = 1;
   piofs_change_view.Hbs = 1;
   piofs_change_view.Hn  = number_of_procs;
   piofs_change_view.subfile = my_proc_num;
   piofs_change_view.flags = NORMAL;
   if ( piofsioctl( fd, PIOFS_CHANGE_VIEW, &piofs_change_view ) != 0 )
      exit( errno );

   /* Initialize the random number generator with a seed based on the
      process number. Fill the block with random numbers then write it
      to the subfile. */
   srand( my_proc_num + 1 );

   /* Loop to write blocks of random numbers into the subfile. */
   for ( i=0; i<NUMBER_OF_BLOCKS; i++ ) {
      /* Loop to fill up a block with random numbers. */
      for ( j=0; j<BLOCK_SIZE; j++ )
         block[ j ] = rand();
      /* Write the block. */
      if ( write( fd, block, BLOCK_SIZE * sizeof(int) ) != BLOCK_SIZE *
sizeof(int) )
         exit( errno );
   }
}
```

Row Partitioning

In row partitioning, each subfile consists of stripes of data that cross every cell as shown in Figure 17.

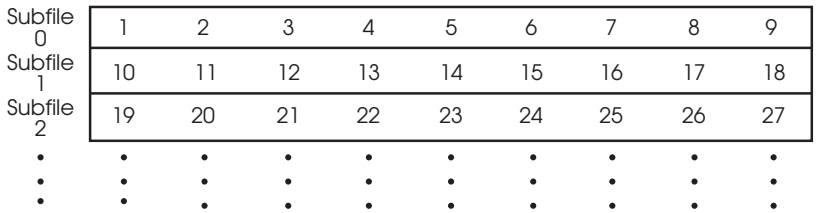

Figure 17. A PIOFS file partitioned by row. Three subfiles containing stripes of data from each cell.

The stripes of each subfile are interleaved with the stripes of all the other subfiles. This is a good way to partition a file in any of the following cases:

- You cannot control the number of cells in the file.
- You want to run programs with different numbers of processes against the same file.
- You have far fewer processes in your program than there are server nodes in the file system so creating one subfile (and cell) per process would not distribute the data well across the servers.

Example of Row Partitioning

To show you how to partition a PIOFS file by row, we've taken the program from the column partitioning example and changed it to partition the PIOFS file across cells instead. The program uses the **piofsioctl()** function call with the PIOFS_STATFS option to determine the number of server nodes and then partitions the PIOFS as follows:

```
# of cells = # of server nodes
# of subfiles = # of processes


/* Simple program to fill a large file with random integers. */
#include <piofs/piofs_ioctl.h>
#include <errno.h>
#include <stdlib.h>
#include <mpproto.h>
#define BLOCK_SIZE 500
#define NUMBER_OF_BLOCKS 1000
main () {
   int my_proc_num, number_of_procs, fd, i, j;
   int block[BLOCK_SIZE];
   int allgrp, nbuf[4];
   piofs_create_t piofs_create;
   piofs_change_view_t piofs_change_view;
   piofs_statfs_t piofs_statfs;

   /* Determine the total number of processes and the number of this
      process. my_proc_num is a number in the range 0 to
      number_of_procs - 1, which we will later use to specify what
      subfile to open. */
   if ( mpc_environ( &number_of_procs, &my_proc_num ) != 0 )
      exit( mperrno );

   /* Obtain the group ID of the group that includes all processes. */
   if ( mpc_task_query( nbuf, 4, 3 ) != 0 )
      exit( -1 );
   allgrp = nbuf[ 3 ];

   /* Find out how many server nodes there are. */
   strcpy (piofs_statfs.name,"/piofs");
   if ( piofsioctl( 0, PIOFS_STATFS, &piofs_statfs ) != 0 )
      exit( errno );

   /* Process 0 creates the PIOFS file to hold the random numbers. */
   if ( my_proc_num == 0 ) {
      /* Create the PIOFS output file with the number of cells equal
         to the number of server nodes. */
      strcpy (piofs_create.name,"random_int_output_file");
      piofs_create.bsu  = BLOCK_SIZE * sizeof( int );
      piofs_create.cells = piofs_statfs.f_nodes;
      piofs_create.permissions = 0644;
      piofs_create.base_node = -1;
      piofs_create.flags = 0;

      if ( piofsioctl( 0, PIOFS_CREATE, &piofs_create ) != 0 )
         exit( errno );
   }
```

```
   /* Synchronize all processes before opening the file to ensure that
      the file has been successfully created before we issue any open
      requests. */
   if ( mpc_sync( allgrp ) )
      exit( -1 );

   /* Open the file to get a file descriptor. */
   fd = open( "random_int_output_file", O_WRONLY, 0 );
   if ( fd == -1 )
      exit( errno );

   /* Change the view of the file so that each process has its own
      subfile. Each subfile contains stripes of data from all cells. */
   piofs_change_view.Vbs = 1;
   piofs_change_view.Vn  = number_of_procs;
   piofs_change_view.Hbs = 1;  /* This could also be piofs_statfs.fnodes */
   piofs_change_view.Hn = 1;
   piofs_change_view.subfile = my_proc_num;
   piofs_change_view.flags = NORMAL;
   if ( piofsioctl( fd, PIOFS_CHANGE_VIEW, &piofs_change_view ) != 0 )
      exit( errno );

   /* Initialize the random number generator with a seed based on the
      process number. Fill the block with random numbers then write it
      to the subfile. */
   srand( my_proc_num + 1 );

   /* Loop to write blocks of random numbers into the subfile. */
   for ( i=0; i<NUMBER_OF_BLOCKS; i++ ) {
      /* Loop to fill up a block with random numbers. */
      for ( j=0; j<BLOCK_SIZE; j++ )
         block[ j ] = rand();
      /* Write the block. */
      if ( write( fd, block, BLOCK_SIZE * sizeof(int) ) != BLOCK_SIZE *
sizeof(int) )
         exit( errno );
   }
}
```

Note that both the column and the row partitioning examples could have been developed by minimally modifying an ordinary serial program that reads or writes to one or more files. In many cases, modifying a serial program to create and partition a PIOFS file can be easier than other alternatives such as:

- Dividing up a regular AIX file using different offsets into the file
- Creating a separate file for each process in the application.

Example of Sorting Data using PIOFS

Here is an implementation of the sorting example presented at the beginning of the chapter.

To perform a sort quickly, you can dynamically partition the file into multiple subfiles and sort each subfile in parallel using a number of different algorithms. For example, you can map the flat file data onto a two-dimensional mesh and then use any sorting algorithm that works on a two-dimensional mesh. For this example, we

use a two-dimensional sort algorithm called a Shear sort. There are other faster algorithms for sorting that could use the same or similar views to divide the file as were used here for the Shear sort. Our purpose, however, is to illustrate the use of PIOFS file partitioning so we have chosen a sort with simple, easy-to-follow steps.

In this example, a file of integers is sorted. To sort the file in this manner, two PIOFS views are used to logically organize the file data. One view partitions the file by row, the other view partitions the file by column. Each process works on only one subfile at a time but the assigned subfile alternates between one in the row view and one in the column view.

Within each subfile, the integers are grouped into smaller units called blocks. To sort its subfile, each process does the following:

- Sorts the numbers in each block.
- For a number of interations equal to the logarithm base 2 of the number of rows of the file:
  - Uses the row view to sort the even rows in increasing order and the odd rows in decreasing order
  - Uses the column view to sort the columns in increasing order.
- Uses the row view to sort each row in increasing order.

Figure 18 illustrates the procedure. This figure is the same as the one presented earlier.

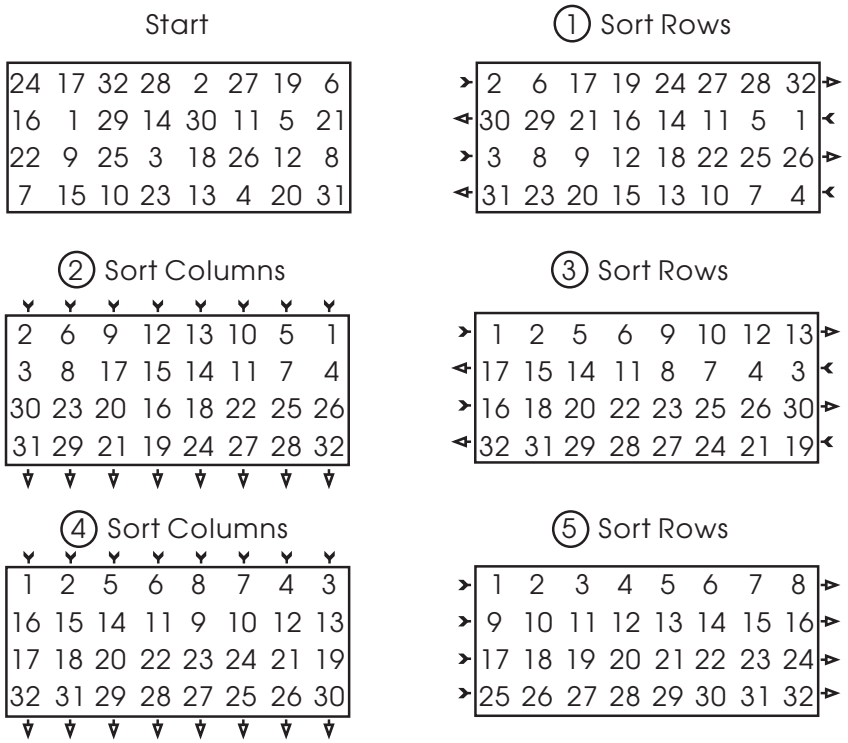

Figure 18. A Shear sort using a partitioned PIOFS file.

The Shear sort program below uses subroutines that are part of the IBM AIX Parallel Environment program product but you can substitute equivalent functions.

```
/* Parallel program to sort a file of integers using a Shear sort */
#include <piofs/piofs_ioctl.h>
#include <errno.h>
#include <stdlib.h>
#include <mpproto.h>
#define BLOCK_SIZE 500
#define NUMBER_OF_BLOCKS 5000
#define INCREASING 1
#define DECREASING 0
main () {
   int my_proc_num, number_of_procs, fd_col, fd_row;
   int sort_direction, row_length, column_length;
   int log2_of_rows, temp;
   int allgrp, nbuf[4];
   piofs_fstat_t piofs_fstat;
   piofs_change_view_t piofs_change_view;

   /* Determine the total number of processes and the number of this
      process. my_proc_num is a number in the range 0 to
      number_of_procs - 1, which we will later use to specify what
      subfiles to open. */
   if ( mpc_environ( &number_of_procs, &my_proc_num ) != 0 )
      exit( mperrno );

   /* Check that the number of blocks is a multiple of the number of
      processes. */
   if ( NUMBER_OF_BLOCKS % number_of_procs != 0 )
      exit( -1 );

   /* Get the group id of the group that includes all processes. */
   if ( mpc_task_query( nbuf, 4, 3 ) != 0 )
      exit( -1 );

   allgrp = nbuf[ 3 ];

   /* Open the file twice to get two file descriptors. */
   fd_col = open( "file_to_sort", O_RDWR, 0 );
   fd_row = open( "file_to_sort", O_RDWR, 0 );

   /* Check that the file has the right number of cells. */

   if ( piofsioctl( fd_col, PIOFS_FSTAT, &piofs_fstat ) != 0 )
      exit( errno );

   if ( piofs_fstat.cells != number_of_procs )
      exit( -1 );

   /* Check that the file has the right BSU size.  This is
      necessary to ensure that the rows are of the correct depth. */
   if ( piofs_fstat.bsu != BLOCK_SIZE * sizeof( int ) )
      exit( -1 );

   /* Now change the views to get both a column and a row view.
      This will modify each file descriptor so that it will provide
      access to the subfile we want. */
   piofs_change_view.Vbs = 1;
   piofs_change_view.Vn  = 1;
```

```
piofs_change_view.Hbs = 1;
piofs_change_view.Hn  = number_of_procs;
piofs_change_view.subfile = my_proc_num;
piofs_change_view.flags = NORMAL;
if ( piofsioctl( fd_col, PIOFS_CHANGE_VIEW, &piofs_change_view ) != 0 )
   exit( errno );

/* Note that the rows are one BSU deep, and that the row view will
   wrap around to eventually cover the entire file. */
piofs_change_view.Vbs = 1;
piofs_change_view.Vn  = number_of_procs;
piofs_change_view.Hbs = 1;
piofs_change_view.Hn  = 1;
piofs_change_view.subfile = my_proc_num;
piofs_change_view.flags = NORMAL;
if ( piofsioctl( fd_row, PIOFS_CHANGE_VIEW, &piofs_change_view ) != 0 )
   exit( errno );

/* Get the length in blocks of each row and column and determine the
   sorting direction for most of the row sorts. */
column_length = NUMBER_OF_BLOCKS / number_of_procs;
row_length = number_of_procs;
sort_direction = (my_proc_num & 1) ? DECREASING : INCREASING;

/* Call a function that sorts all the data within each block of the
   subfile. The subfile is then written back.  */
sort_blocks_of_subfile( fd_col, BLOCK_SIZE, column_length );

/* Iterate between sorting the rows in opposite directions, and
   sorting the columns in increasing order.  The iteration is
   repeated a number of times equal to the next integer greater
   than or equal to the logarithm base 2 of the number of rows in
   the file.  First, determine the number of iterations. */
for ( log2_of_rows = 0, temp = 1;
      temp < column_length;
      log2_of_rows++, temp <<= 1 );

for ( ; log2_of_rows > 0; log2_of_rows-- ) {

   /* Call a function that sorts the data within each row subfile
   of the file.  Each row consists of a number of presorted
   blocks, one from each cell of the file.  Since number of rows
   is a multiple of the number of processes, each row subfile can
   include several rows interleaved into the file. We specify a
   window size equal to the size of one row, in which the presorted
   blocks are sorted. Blocks are not sorted across the boundary of
   the window size within each subfile. We first determine the
   sorting direction to use.  All of the even rows will be sorted
   into increasing order, and all of the odd rows will be sorted
   into decreasing order. */
   sort_subfile( fd_row, BLOCK_SIZE, row_length, sort_direction );

   /* Synchronize with all the other processes before sorting
      along the columns. */
   if ( mpc_sync( allgrp ) != 0 )
      exit( -1 );

   /* Call the same function to completely sort each column of the
```

```
      file. */
      sort_subfile( fd_col, BLOCK_SIZE, column_length, INCREASING );

      /* Synchronize with all the other processes before sorting along
      the rows. */
      if ( mpc_sync( allgrp ) != 0 )
         exit( -1 );
   }

   /* Finally, sort all the rows into increasing order.  */
   sort_subfile( fd_row, BLOCK_SIZE, row_length, INCREASING );

   /* Now, the entire file is sorted. */
   close( fdcol );
   close( fdrow );
}
```

Example of Performing a Matrix Multiplication with PIOFS

Here is an implementation of the matrix multiplication example presented at the beginning of the chapter.

The program computes the product, C, of two square matrices A and B, which are stored in PIOFS files, matA and matB. The resulting matrix is then stored in PIOFS file, matC. Files matA and matB are assumed to exist prior to the run.

Four compute tasks are running in this parallel application. If N is the dimension of the matrices:

- Tasks 0 and 1 process the first N/2 rows of matrix A
- Tasks 2 and 3 process the last N/2 rows of matrix A
- Tasks 0 and 2 process the first N/2 columns of matrix B
- Tasks 1 and 3 process the last N/2 columns of matrix B

Therefore, each task computes a quadrant of the resulting matrix. N is assumed to be a multiple of 2, and each file is formed of N cells, each cell containing a column of the matrix stored in the file. The BSU size of the files is one matrix element. Matrix elements are integers. Figure 19 illustrates the procedure. This figure is the same as the one presented earlier.

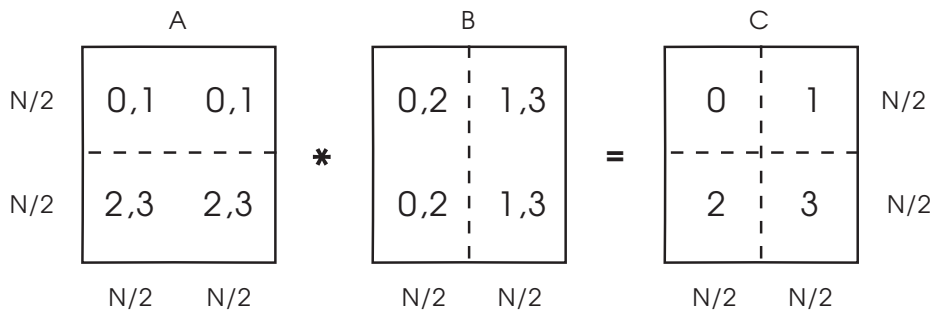

Figure 19. A matrix multiplication using partitioned PIOFS files. Four compute tasks perform the operation.

This program uses subroutines that are part of the IBM AIX Parallel Environment program product but you can substitute equivalent functions.

```
/* Parallel program to perform a matrix multiplication */
#include <piofs/piofs_ioctl.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
#include <mpproto.h>

#define  N       16
#define  BLK    N/2

main(int argc, char **argv)
{
   int                  i, j, k;      /* loop indices */
   int                  nprocs;       /* number of processes attached to the run */
   int                  myprocid;     /* process id */
   int                  allgrp;       /* id of group including all processes */
   int                  nbuf[4];      /* buffer for task query */
   int                  A[N][BLK];    /* local matrix A */
   int                  B[BLK][N];    /* local matrix B */
   int                  C[BLK][BLK];  /* local matrix C */
   char                *matA;         /* file containing matrix A */
   char                *matB;         /* file containing matrix B */
   char                *matC;         /* file containing matrix C */
   int                  fdA;          /* file descriptor */
   int                  fdB;          /* file descriptor */
   int                  fdC;          /* file descriptor */
   int                  rc;           /* return code */
   int                  nb;           /* number of bytes read or written */
   piofs_change_view_t  myview;       /* structure for changing view */
   piofs_create_t       mycreat;      /* structure for creating file matC */

   /* Get file names */
   matA = argv[1];
   matB = argv[2];
   matC = argv[3];

   /* Get process id (program is assumed to run under POE) */
   if ( mpc_environ( &nprocs, &myprocid ) != 0 )
      exit( mperrno );

   /* Get id of group including all processes */
   if ( mpc_task_query( nbuf, 4, 3 ) != 0 )
      exit( -1 );
   allgrp = nbuf[3];

   /* Open files matA and matB in read mode */

   if ( ( fdA = open( matA, O_RDONLY ) ) == -1 )
      exit( errno );
   if ( ( fdB = open( matB, O_RDONLY ) ) == -1 )
      exit( errno );

   /* Define partitioning for matrices A and B */

   myview.Vbs = BLK;        /* one horizontal slice of matrix A of  */
   myview.Vn = 2;           /* size BLK is assigned to each process */
```

```
myview.Hbs = N;
myview.Hn = 1;
myview.subfile = myprocid / 2;
myview.flags = ACTIVE|NORMAL;
if ( ( rc = piofsioctl( fdA, PIOFS_CHANGE_VIEW, &myview ) ) != 0 )
   exit( errno );

myview.Vbs = N;          /* one vertical slice of matrix B of    */
myview.Vn = 1;           /* size BLK is assigned to each process */
myview.Hbs = BLK;
myview.Hn = 2;
myview.subfile = myprocid % 2;
myview.flags = ACTIVE|NORMAL;
if ( ( rc = piofsioctl ( fdB, PIOFS_CHANGE_VIEW, &myview ) ) != 0 )
   exit( errno );

/* Read local matrices A and B */

if ( ( nb = read( fdA, A, N*BLK*sizeof( int ) ) ) != N*BLK*sizeof( int ) )
   exit( errno );

if ( ( nb = read( fdB, B, BLK*N*sizeof( int ) ) ) != BLK*N*sizeof( int ) )
   exit( errno );

/* Close files matA and matB */

if ( ( rc = close( fdA ) ) != 0 )
   exit( errno );
if ( ( rc = close( fdB ) ) != 0 )
   exit( errno );

/* Perform local matrix product */

for ( i=0 ; i<BLK ; i++ ) {
   for ( j=0 ; j<BLK ; j++ ) {
      C[i][j] = 0;
      for ( k=0 ; k<N ; k++ ) {
         C[i][j] += A[k][j] * B[i][k];
      }
   }
}

/* Create file matC (only task 0 creates file) */

if ( myprocid == 0 ) {
   strcpy( mycreat.name, matC );
   mycreat.bsu = sizeof( int );
   mycreat.cells = N;
   mycreat.permissions = 0600;
   mycreat.base_node = -1;
   mycreat.flags=REPLACE;
   if ( ( rc = piofsioctl( 0, PIOFS_CREATE, &mycreat ) ) != 0 )
      exit( errno );
}
mpc_sync( allgrp );             /* synchronize with other tasks */

/* Open file matC in write mode */
```

```
   if ( ( fdC = open( matC, O_WRONLY ) ) == -1 )
      exit( errno );

   /* Define partitioning for matrix C */

   myview.Vbs = BLK;              /* one quadrant of matrix C of size */
   myview.Vn = 2;                 /* BLK is assigned to each process  */
   myview.Hbs = BLK;
   myview.Hn = 2;
   myview.subfile = myprocid;
   myview.flags = ACTIVE|NORMAL;
   if ( ( rc = piofsioctl( fdC, PIOFS_CHANGE_VIEW, &myview ) ) != 0 )
      exit( errno );

   /* Write local matrix C */

   if ( ( nb = write( fdC, C, BLK*BLK*sizeof( int ) ) ) != BLK*BLK*sizeof( int
) )
      exit( errno );

   /* Close file matC */

   if ( ( rc = close( fdC ) ) != 0 )
      exit( errno );

   mpc_sync( allgrp );            /* synchronize with other tasks     */
}
```

Chapter 8. Programming Considerations

This chapter contains information about:

- Understanding Atomicity of Reads and Writes
- Controlling concurrent access to files
- Maximizing PIOFS I/O Performance
- Checkpointing files
- Specifying file attributes for new Parallel I/O File System files
- Accessing PIOFS files correctly
- Using an Existing, Unchanged Program with a PIOFS File

Understanding Atomicity of Reads and Writes

PIOFS processes reads and writes atomically. That is, once a read or write function begins execution it completes as if no other read or write were active in the Parallel I/O File System during the time of its execution. So, with atomicity, no read would return a portion of data being written by a concurrent write operation, even on another client node.

Atomic does not mean that a read or write must complete entirely or not at all. The Parallel I/O File System does not guarantee all or none. For example, when writing to a subfile that spans multiple cells, if the request is not fully satisfied, you do not know which data has been written.

Controlling Concurrent Access to Parallel I/O File System Files

Typically, an application is designed to ensure that two tasks cannot access the same data at the same time incompatibly. With the Parallel I/O File System, you have the option of letting the Parallel I/O File System manage concurrent access for you if your application doesn't handle this responsibility itself.

The Parallel I/O File System lets you open a subfile in either of two ways — NORMAL or CAUTIOUS. If you choose NORMAL, the Parallel I/O File System does nothing to serialize concurrent access. NORMAL is appropriate when either:

- Your application has its own mechanism for serializing concurrent access
- Each task is accessing different subfiles. Subfiles created from the same view are always disjoint. There can never be a conflict between two tasks reading or writing different subfiles from the same view.

Write processing is usually faster with the NORMAL option than with CAUTIOUS option, so it makes sense to choose it if you can.

Most programs will want to do their own management of overlapping writes. For this reason and because write processing can be slightly slower with CAUTIOUS, NORMAL is the default.

If your application does not manage concurrent data access, you should choose CAUTIOUS. With CAUTIOUS, the Parallel I/O File System guarantees that each read or write is completed atomically.

© Copyright IBM Corp. 1995

For instance, suppose the same data is mapped to two different subfiles (known as overlapping subfiles) because two tasks have opened the same file using two different views. If the common data spans multiple cells and both tasks update the common data simultaneously, the resulting data could potentially include partial updates from each task unless both subfiles have been opened with CAUTIOUS or the application has its own method of controlling concurrent writes.

"Examples of Using NORMAL" shows the results of using NORMAL with and without the management of concurrent writes by the application.

If CAUTIOUS is specified, each task's updates are completed before the other task's updates are allowed to proceed.

Setting the CAUTIOUS or NORMAL Flag

To ensure compatibility with the existing **open( )** function, the **open( )** function flags are not used for any new purpose in the Parallel I/O File System. Any new Parallel I/O File System function is done with the **piofsioctl()** function with the PIOFS_CHANGE_VIEW option. First a process opens the file, then calls the **piofsioctl()** function with the PIOFS_CHANGE_VIEW option to set the NORMAL or CAUTIOUS flag.

The Parallel I/O File System default is NORMAL. To change the default, you can add the keyword CAUTIOUS to the **piofs.profile** as described on page 89.

Examples of Using NORMAL

The following examples show what might happen when you specify NORMAL in specific situations. In the first example, two tasks write to a subfile without attempting to coordinate with each other. In the second example, Task 2 waits for a message from Task 1.

NORMAL Without Program Control Over the Order of Writes

In this example, shown in Figure 20 on page 87, there is one subfile, with four cells numbered 0 through 3. The view is:

Vbs = 1
Hbs = 4
Hn = 1
Vn = 1

There are two tasks; Task 1 and Task 2. Each writes four BSUs of data to the same place.

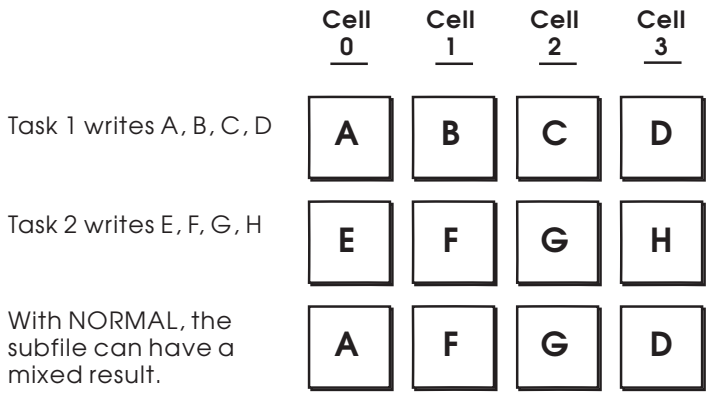

Figure 20. Write operations with no program control over the order of writes. Task 1 writes A, B, C, D. Task 2 writes E, F, G, H. After the writes, the subfile could contain mixed data if NORMAL is used.

NORMAL With Program Control Over the Order of Writes

In this example, Figure 21, there is one subfile, with four cells numbered 0 through 3. The view is the same as in Figure 20:

Vbs = 1
Hbs = 4
Hn = 1
Vn = 1

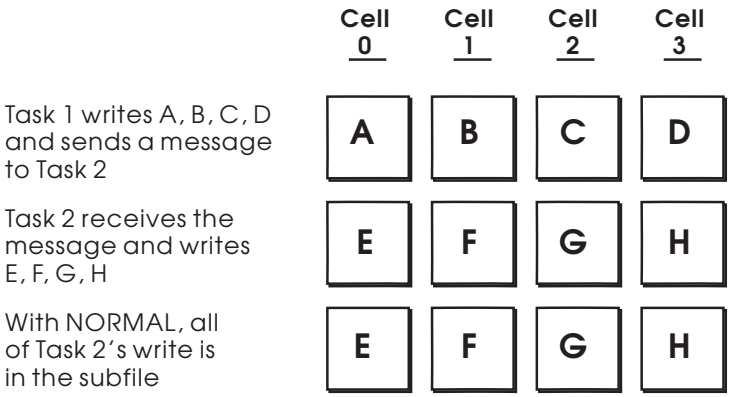

Figure 21. Write operations with program control over the order of writes. Task 1 writes A, B, C, D, then sends a message to Task 2 which writes E, F, G, H. After both writes, Task 2's data is in the subfile.

Because Task 1's write is synchronous, Task 1 does all its writes before sending its message to Task 2. If the program controls the order of writes, there is no need to use the CAUTIOUS flag. If you are not doing overlapping writes, the NORMAL flag is sufficient to achieve atomicity.

Maximizing PIOFS I/O Performance

To obtain the best I/O performance for your application program, perform file input and output operations using the **read()** and **write()** system calls. Using these system calls minimizes the number of times that data must be copied between buffers.

You can also maximize I/O performance by ensuring that the amount of data read or written at a time is:

- A multiple of 64K bytes
- At least 256K bytes.

Below are performance tips geared specifically to C language and FORTRAN application programs.

C Language Programs

Using buffered I/O functions such as **fwrite()** and **fread()**, which are part of the C Standard I/O Library, can cause significant performance degradation. These I/O functions use a default buffer size of only 4096 bytes for I/O operations, which limits the amount of data that can be read or written at a time by a Parallel I/O File System. This constraint occurs regardless of the amount of data specified in the **fwrite()** or **fread()** function call.

To alleviate this problem, you can invoke the **setvbuf()** function to create a larger buffer, enabling the Parallel I/O File System to read and write larger amounts of data at time. Issue **setvbuf()** after opening the file, but before reading or writing any data to it.

For example, to set the buffer size to WRITE_SIZE, where WRITE_SIZE is defined as 1048576:

```
#define WRITE_SIZE 1048576

FILE *fp;

cptr = (char *)malloc(WRITE_SIZE);
setvbuf(fp, cptr, _IOFBF, WRITE_SIZE);
```

For more information about **setvbuf()**, see the setvbuf man page.

Please also note that if you use **fread** and **fwrite** and also wish to use a **piofsioctl** option such as PIOFS_LSEEK, you must obtain the file descriptor corresponding to the file pointer used for **fread** or **fwrite**. This can be done using the AIX fileno() macro. Refer to the fileno man page for more detail.

FORTRAN Programs

To obtain the best I/O performance for your FORTRAN application program, open your files for direct access not sequential access and set the record length to be:

- A multiple of 64K bytes
- At least 256K bytes.

Also, specify the amount of data to write or read at compile time rather than execution time. For example, specify:

```
WRITE(11) buf
```

instead of

```
WRITE(11) buf(1:n)
```

Checkpointing Files

Use the PIOFS checkpointing function to create a snapshot of a file's contents at a particular point during processing. If, for some reason, your program fails, you may be able to restart the program using the file's contents at that checkpoint. You can also use a checkpoint to preserve a particular version of the file.

To checkpoint a Parallel I/O File System file, use the **piofsioctl()** function call with the PIOFS_CHKPOINT option. See “piofs_chkpoint” on page 105 for the syntax of the PIOFS_CHKPOINT option.

All changes in a file after the file has been checkpointed are made to new blocks of storage. The old storage blocks are saved unchanged. At any one time, there can be either:

- A single, active version of a file
- Both the active file and the checkpointed version of the file.

When a file is deleted, both versions are destroyed.

You cannot modify the checkpointed version of a file. You can, however, delete it using the **piofsioctl()** function call with the PIOFS_DELETE_CHKPOINT option. The active version of the file can be rolled back to the checkpointed version by **piofsioctl()** with the PIOFS_RESTORE_CHKPOINT option.

When you create a checkpoint, make sure that there is enough space in the Parallel I/O File System to hold a complete copy of both the active and checkpointed versions of the file. The active version requires additional space only for those blocks that are changed.

To make a copy of the checkpointed version (as opposed to the active version), open it and use **piofsioctl()** with the PIOFS_CHANGE_VIEW option to access the file.

Specifying File Attributes for Parallel I/O File System Files

You can create a Parallel I/O File System file using any of the following:

- The **open()** function
- The PIOFS_CREATE option of the **piofsioctl()** function call (C language)
- The PIOFSCREATE function (FORTRAN).

When you create a file using either the PIOFS_CREATE option or the PIOFSCREATE function, you specify the file attributes you want using parameters.

When you create or open a file using **open** (instead of using a Parallel I/O File System function call) and a **piofs.profile** file exists, your file is assigned the attributes specified in the **piofs.profile** file.

You can use the **piofs.profile** file to specify all or only some file attributes. If you don't use a **piofs.profile** or if you omit some file attributes from the **piofs.profile** file, the default values listed in “Default File Attributes” on page 91 are used. You can also request the default value for a file attribute by specifying -1 as its value in **piofs.profile**.

The **piofs.profile** file can specify values for the following attributes:

- The size of basic striping unit (bsu)
- The number of cells in the file
- The file offset to use for the subfiles — the AIX system file offset or the PIOFS 64-bit file offset.
- Whether the Parallel I/O File System should force sequential atomicity.
- The base node (the server node at which a file's first cell is to be located). Make sure you don't specify the same base node in each **piofs.profile** because this is likely to cause contention for some server nodes.

The system uses the first **piofs.profile** file found by searching as follows:

1. **$HOME**/*file_name*.**piofs.profile**, which is the local default to be applied to *file_name* when you access that file.
2. **$HOME**/**piofs.profile**, which is the local default to be applied to all files.
3. /**usr**/**lpp**/**piofs**/**piofs.profile**, which is to be applied to all files on the client node. To be effective on all client nodes, all client nodes must be able to access this file.

A Sample piofs.profile File

The **piofs.profile** file has the following syntax:

- Comments begin with a pound sign (#).
- Extra blanks and tabs in a line are ignored. Separate tokens with a space.
- The keywords are case sensitive and must be upper case.

```
# This is a profile for the IBM Parallel I/O File System.
# It gets read during file create and open time.  The following
# types of records are supported:
# 1) # comment
# 2) BSU              Specifies the basic striping unit in bytes. Specify
#                     -1 to use the default value of 32,768 bytes.
#                     The maximum is 2,147,483,647 bytes.
# 3) CELLS            Specifies the number of cells to use
#                     when creating the file. Specify -1 to use the
#                     default value of one per server node.
# 4) USE_PIOFS_LSEEK  Indicates that 64-bit offsets are to be used
#                     instead of the AIX file offset.
# 5) CAUTIOUS         Indicates that the file system should
#                     force sequential atomicity.
# 6) BASE_NODE        Specifies the node at which cell 0 is to be located
#                     We recommend that you specify -1 to use the
#                     default, which is pseudo-random base node
#                     selection. If you choose to specify a base node,
#                     do not specify it in /usr/lpp/piofs/piofs.profile,
#                     because that would cause all files to start at
#                     the same node creating a likely bottleneck.
#

#
# Uncomment the following lines to use this as a skeleton
# BSU                 32768
# CELLS               4
# USE_PIOFS_LSEEK
# CAUTIOUS
# BASE_NODE           -1
```

Default File Attributes

If no **piofs.profile** exists, Parallel I/O File System files are created with the following defaults:

- A basic striping unit equal to 32,768 bytes
- The number of cells is one per server node
- The file offset is the standard AIX file offset
- The default view is *Vbs* = *Hbs* = *Vn* = *Hbs* = 1. Thus the default is to have one subfile, which is treated logically as a single AIX file.
- The *base node* is pseudo-randomly selected by the file system. This default is the recommended method of setting the base node.

Accessing PIOFS Files Correctly

To access PIOFS files correctly, you need to understand when the AIX file offset is valid and when it is not. Some AIX functions, such as **lseek** or functions that use **lseek**, require a valid AIX file offset to work. However, once you begin to use the PIOFS file offset, (as described later in this section), the AIX file offset is no longer used by PIOFS and can contain the wrong offset value.

This section explains how to access PIOFS files and subfiles under different circumstances and how to ensure that you use a valid file offset.

The appropriate methods for accessing a PIOFS file or subfile are determined by the size of the PIOFS file or subfile, the programming language, and the version of AIX. Table 4 on page 93 provides specific information for accessing PIOFS file or subfiles 2GB or larger under different circumstances.

Files Smaller Than Two Gigabytes

For PIOFS files or subfiles smaller than 2GB, the AIX file offset is valid **unless** you have begun using the PIOFS file offset (see Table 4 on page 93).

If you have not begun using the PIOFS file offset, you can safely use any supported AIX or PIOFS functions involving file access because the AIX file offset contains a valid value.

If your program never accesses files or subfiles 2GB or larger, there is no reason to use the PIOFS file offset or the PIOFS file access functions. In fact, we recommend that you use only the AIX file offset and AIX file access functions.

If you do begin using the PIOFS file offset, the AIX file offset becomes invalid and you can no longer use **lseek** or other AIX functions that use **lseek** such as the vi editor.

Files Two Gigabytes or Larger

For PIOFS files or subfiles that are or might be 2GB or larger, you must use the PIOFS file offset. However, as stated previously, once you begin using the PIOFS file offset, the AIX file offset becomes invalid and you can no longer use **lseek** or other AIX functions that use **lseek** such as the vi editor.

To establish the use of the PIOFS file offset, there are several options. Table 4 on page 93 lists the circumstances under which each is appropriate.

Important

If you encounter file offset errors with files 2GB or larger when using FORTRAN, ensure that the system administrator has installed the required PTFs. See “Software Requirements” on page 9 for further information.

Table 4. Correct Ways to Establish Access to Files 2GB or Larger

Programs running on AIX Version 3.2.5			
C program		**FORTRAN program**	
New program	**Existing program**	**New program**	**Existing program**
Either of the following: • Use **piofs.profile** with USE_PIOFS_LSEEK • Issue **piofsioctl** with the PIOFS_LSEEK option. Do this at least once to initiate the use of the PIOFS file offset. "Initiating the Use of the PIOFS Offset Using PIOFS_LSEEK" on page 94 shows you how to do this.	Use **piofs.profile** with USE_PIOFS_LSEEK.	No action necessary. File offset is correctly managed by the system.	Runs correctly if your program is not **statically** bound to the FORTRAN run-time library routines.
Programs running on AIX Version 4.1			
C program		**FORTRAN program**	
New program	**Existing program**	**New program**	**Existing program**
Either of the following: • Use **piofs.profile** with USE_PIOFS_LSEEK • Issue **llseek**. Do this at least once to initiate the use of the PIOFS file offset. "Initiating the Use of the PIOFS Offset Using llseek" on page 94 shows you how to do this.	Use **piofs.profile** with USE_PIOFS_LSEEK.	No action necessary. File offset is correctly managed by the system.	Runs correctly if your program is not **statically** bound to the FORTRAN run-time library routines.

Initiating the Use of the PIOFS Offset Using PIOFS_LSEEK

If you are running a C program under AIX Version 3.2.5 and accessing PIOFS files 2GB or larger, you can use the method shown below to initialize the PIOFS offset. Typically, you would seek to the beginning of the file immediately after opening it. Be sure to check Table 4 on page 93 to verify that this is an appropriate technique for your program.

```
piofs_lseek_t myseek;

fd = open(filename, flags, permissions);
myseek.offset.h = 0;
myseek.offset.l = 0;
myseek.whence = SEEK_SET;
piofsioctl(fd, PIOFS_LSEEK, &myseek);
```

Initiating the Use of the PIOFS Offset Using llseek

If you are running a C program under AIX Version 4.1 and accessing PIOFS files 2GB or larger, you can use the method shown below to initialize the PIOFS offset. Typically, you would seek to the beginning of the file immediately after opening it. Be sure to check Table 4 on page 93 to verify that this is an appropriate technique for your program.

```
fd = open(filename, flags, permissions);
llseek(fd, (offset_t)0, SEEK_SET);
```

Using Existing Programs with a PIOFS File

To use an existing program with a Parallel I/O File System file, you must do the following:

- Create the Parallel I/O File System file as a regular AIX file using **open()**
- Ensure that the existing code uses no AIX subroutines, subroutine parameters, or functions unsupported or modified by the Parallel I/O File System. This means primarily that your program must not use:
  - File locking
  - **open()** with the append option
  - **lseek**.
- If the file is 2GB or larger, create a **piofs.profile** file with the keyword, USE_PIOFS_LSEEK, to indicate that PIOFS should use its internal 64-bit offset. Be sure to see “Accessing PIOFS Files Correctly” on page 92 for important information regarding the appropriate use of the PIOFS file offset.

You must recompile and relink your source code only if you have to change it because it uses unsupported or modified AIX subroutines, subroutine parameters, or functions, **or**, if you want to use PIOFS function calls.

Chapter 9. Using PIOFS Functions and Base AIX Functions

This chapter contain information about:

- Parallel I/O File System C and FORTRAN functions
- Supported AIX functions and any PIOFS-related changes to them

Using PIOFS Functions

The C and FORTRAN functions provided with the Parallel I/O File System are described below. When you invoke a program containing calls to PIOFS functions, bear in mind that the program can be loaded only on nodes that have a PIOFS installed and mounted. This is because all PIOFS function calls must be resolved regardless of whether they will be executed.

If you are using the AIX Parallel Environment program product, remember that if you invoke a program without using the **poe** command, the program is loaded on the node from which you issued the invocation even when the program will actually run on another node. Therefore, the node from which you invoke the program must have a PIOFS installed and mounted.

PIOFS Functions for C Language

Most application programming for the Parallel I/O File System is done through **piofsioctl()** functions which are analogous to the **ioctl** functions used for other files and devices. The **piofsioctl()** functions have the same syntax, but have operations designed for parallel files. Once a file is defined, either using the Parallel I/O File System's unique functions or by opening it with the Parallel I/O File System's default parameters, your application can call the usual **read**, **write**, **open**, and **close** functions as described in “Supported AIX Functions” on page 97.

© Copyright IBM Corp. 1995

The **piofsioctl()** function supports the following operations:

Table 5. Parallel I/O File System PIOFS C Language Functions	
piofsioctl() Function	**Purpose**
PIOFS_CHANGE_VIEW	To access a different subfile. See page 101.
PIOFS_CHKPOINT	To checkpoint a PIOFS file. See page 105.
PIOFS_CREATE	To create a Parallel I/O File System file. See page 107.
PIOFS_DELETE_CHKPOINT	To delete the checkpointed version of a file. See page 110.
PIOFS_FSTAT	To retrieve the status record from a Parallel I/O File System file. See page 112.
PIOFS_LSEEK	To set or determine the offset in a Parallel I/O File System file. See page 115.
PIOFS_RESTORE_CHKPOINT	To restore the checkpointed version of a Parallel I/O File System file. See page 118.
PIOFS_STATFS	To get system status information for all Parallel I/O File System servers. See page 120.

Parallel I/O File System FORTRAN Functions

The Parallel I/O File System has the following new functions for FORTRAN:

Table 6. Parallel I/O File System FORTRAN Language Functions

Function	Purpose
PIOFSCHANGEVIEW	To access a different subfile. See page 101.
PIOFSCHKPT	To checkpoint a PIOFS file. See page 105.
PIOFSCREATE	To create a Parallel I/O File System file. See page 107.
PIOFSDELCHKPT	To delete a checkpointed version of a file. See page 110.
PIOFSFSTAT	To retrieve the status record from a Parallel I/O File System file. See page 112.
PIOFSRCHKPT	To restore the checkpointed version of a Parallel I/O File System file. See page 118.
PIOFSSTATFS	To get system status information for all Parallel I/O File System servers. See page 120.

Note: There is no FORTRAN version of the PIOFS_LSEEK **piofsioctl()** option.

In addition to the specific Parallel I/O File System functions, FORTRAN statements are also supported for AIX Version 4.1 For AIX Version 3.2.5, they are supported **but only for subfiles less than 2 GB in size**.

Supported AIX Functions

The Parallel I/O File System supports some AIX functions that affect files with no changes or restrictions. Several functions, however, have restrictions, extensions, or minor limitations.

Base AIX File-Handling Subroutines Affected By the Parallel I/O File System

Each function or subroutine that is affected by the Parallel I/O File System is described by a reference page as follows:

Table 7. AIX Subroutines Affected by the Parallel I/O File System	
Subroutine	**Where Information Can Be Found**
access	“access, accessx, or faccessx” on page 126
chmod	“chmod or fchmod” on page 129
chown	“chown, fchown, chownx, or fchownx” on page 133
close	“close” on page 137
fclose	“fclose or fflush” on page 139
fcntl	“fcntl, dup2, or dup” on page 141
fopen	“fopen, freopen, or fdopen” on page 149
fread	“fread or fwrite” on page 154
fseek	“fseek, rewind, ftell, fgetpos, or fsetpos” on page 157
getc	“getc, getchar, fgetc, or getw” on page 161
llseek	“llseek (AIX Version 4.1)” on page 164
lseek	“lseek” on page 166
mkdir	“mkdir” on page 168
open	“open, openx, creat” on page 171
putc	“putc, putchar, fputc, or putw” on page 179
puts	“puts or fputs” on page 182
read	“read, readx, readv, or readvx” on page 184
remove	“remove” on page 190
rename	“rename” on page 191
rmdir	“rmdir” on page 194
stat	“stat, lstat, fstat, statx, fstatx, fullstat, or ffullstat” on page 197
unlink	“unlink” on page 202
write	“write, writex, writev, or writevx” on page 204

Base AIX File-Handling Subroutines Unaffected By PIOFS

The following subroutines that relating to files are not affected by the Parallel I/O File System. They have the same effect on Parallel I/O File System files as they do on other AIX files.

chdir
getcwd
umask

Base AIX File-Handling Subroutines Not Supported By PIOFS

The file system helpers,

chfs
fsck
mkfs
rmfs

are not supported. The Parallel I/O File System configuration scripts provide equivalent functions for each of these.

Parallel I/O File System Programming Reference

Programming Information for PIOFS Functions 100
piofs_change_view 101
piofs_chkpoint 105
piofs_create 107
piofs_delete_chkpoint 110
piofs_fstat 112
piofs_lseek 115
piofs_restore_chkpoint 118
piofs_statfs 120

© Copyright IBM Corp. 1995

Programming Information for PIOFS Functions

Header Files

Header files are available for use with the Parallel I/O File System programming functions. The files are linked to the /**usr**/**include**/**piofs** directory. For C programs include the header files with:

```
#include <piofs/piofs_ioctl.h>
```

For FORTRAN programs include the header file with:

```
INCLUDE 'piofs/piofs.inc'
```

Important

The variable type, offset_t, which is defined in **piofs**/**piofs.h**, differs between PIOFS for AIX Version 3.2.5 and PIOFS for AIX Version 4.1. Be sure to check its definition before coding.

Compiling and Linking FORTRAN Programs

The exported function names need the following flag as part of the link phase when compiling:

-bI:/usr/include/piofs/piofs.exp -L/usr/lib/piofs -lPiofsFortran

For instance:

```
xlf -bI:/usr/include/piofs/piofs.exp -L/usr/lib/piofs -lPiofsFortran myprogram.f
```

Compiling and Linking C Programs

To access the **piofsioctl()** functions, use the following flag:

-bI:/usr/include/piofs/piofs.exp

For instance:

```
cc -o myprogram -bI:/usr/include/piofs/piofs.exp myprogram.c
```

piofs_change_view

Purpose

piofs_change_view – Change the view of a Parallel I/O File System file.

C Syntax

#include <**piofs/piofs_ioctl.h**>

int piofsioctl(
  int *fildes*,
  int PIOFS_CHANGE_VIEW,
  piofs_change_view_t **argument* **);**

Parameters

fildes Specifies a previously opened file descriptor returned by the AIX **open()** or **creat()** function.

PIOFS_CHANGE_VIEW
Specifies the **piofsioctl** operation to be performed on the device: Change the view of a Parallel I/O File System file.

argument A pointer to the *piofs_change_view_t* structure:

```
typedef struct {
  int Vbs;     /* Vertical block size in basic striping units */
  int Vn;      /* Number of vertically interleaved subfiles   */
  int Hbs;     /* Horizontal block size                       */
  int Hn;      /* Number of horizontally interleaved subfiles */
  int subfile; /* Subfile to view                             */
  int flags;   /* ACTIVE (default) or CHECKPOINT              */
               /* NORMAL (default) or CAUTIOUS                */
  ulong_t reserved[8]; /* Reserved by the system              */
} piofs_change_view_t ;
```

The flags field is a logical OR of two values: ACTIVE or CHECKPOINT, and NORMAL or CAUTIOUS. A 0 flag defaults to ACTIVE and NORMAL.

ACTIVE Access the active version of the file. This is the default.

CHECKPOINT Access the checkpoint version of the file.

NORMAL Do not use concurrency control to guarantee that each read or write will be completed atomically. Using NORMAL can often improve the performance of your application. However, you should be sure to understand the trade-offs involved. See “Controlling Concurrent Access to Parallel I/O File System Files” on page 85 for more information.

CAUTIOUS Force sequential atomicity; once a read or write function begins execution, it completes as if no other read or write were active in the Parallel I/O File System while it is executing.

FORTRAN Syntax

INTEGER FUNCTION PIOFSCHANGEVIEW (*unit*, *vbs*, *vn*, *hbs*, *hn*, *subfile*, *flags***)**

INTEGER*4 unit, vbs, vn, hbs, hn, subfile, flags

unit Unit number of the file

vbs Vertical block size in basic striping units

vn Number of vertically interleaved subfiles

hbs Horizontal block size

hn Number of horizontally interleaved subfiles

subfile The subfile to view

flags Specifies which version of the file to view

ACTIVE View the active version of the file. This is the default.

CHECKPOINT View the checkpoint version of the file.

NORMAL Do not use concurrency control to guarantee that each read or write will be completed atomically. NORMAL is faster than CAUTIOUS and may produce the same results. This is the default.

CAUTIOUS Force sequential atomicity; once a read or write function begins execution, it completes as if no other read or write were active in the Parallel I/O File System while it is executing.

Description

The **piofsioctl()** function with the PIOFS_CHANGE_VIEW option changes the logical view of a Parallel I/O File System file opened previously with **open()** or **creat()**. Both **open()** and **creat()** open the file with *Vbs*, *Vn*, *Hbs*, and *Hn* = 1 and one subfile, namely subfile 0. You must use PIOFS_CHANGE_VIEW to establish any other view.

The file is partitioned according to the parameters *Vbs*, *Vn*, *Hbs*, and *Hn* as described in “Partitioning a File into Subfiles” on page 66. Subfiles are numbered from 0 to ($Hn \times Vn$) − 1. The *subfile* parameter, indicating the subfile to be opened, must be in this range.

When the parameter *flag* is set to CHECKPOINT, subsequent reads to the file will be read from the checkpoint version. You cannot write to a checkpoint version.

When the CAUTIOUS flag is set, the Parallel I/O File System forces sequential atomicity. See “Understanding Atomicity of Reads and Writes” on page 85 for a description of the NORMAL and CAUTIOUS flags.

Any call to the PIOFS_CHANGE_VIEW function resets the file pointer for the subfile given in the **piofs_change_view_t** structure to point to the first byte of the subfile.

Refer to the **ioctl()** reference page for general information about the **ioctl()** function.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

If the **pfsioctl()** function fails, **errno** may be set to one of the following values:

EBADF — The *fildes* or *unit* parameter is not a valid open file descriptor.

EINVAL — One of the following is true:

- One of *Vbs*, *Vn*, *Hbs*, or *Hn* is 0 or negative.
- The *subfile* parameter is not between 0 and $Hn \times Vn - 1$.
- The subfile is empty. No empty subfiles are allowed: $Hbs \times (Hn - 1) + 1$ must be $\leq$ number of cells.
- The *flags* parameter is not set to a valid value.

Examples

1. C Example:

   To open the 6th subfile of *myfile*:

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>

int   rt,     /*  return value     */
      fildes; /* file descriptor   */

piofs_change_view_t ViewStruct;
ViewStruct.Vbs = 4;
ViewStruct.Vn  = 2;
ViewStruct.Hbs = 3;
ViewStruct.Hn  = 4;
ViewStruct.subfile = 5;
ViewStruct.flags = (NORMAL | ACTIVE);

fildes = open("/piofs/thisuser/myfile",O_RDWR,0740);

rt = piofsioctl(fildes, PIOFS_CHANGE_VIEW, &ViewStruct);
```

2. FORTRAN Example:

   To open the 6th subfile of *myfile*:

```
      INCLUDE 'piofs/piofs.inc'

      INTEGER*4 VBS, VN, HBS, HN, SUBFILE, RT, IOS, FLAGS

      OPEN(UNIT=7,IOSTAT=IOS,FILE='/piofs/thisuser/myfile',
     1    STATUS='OLD',ACCESS='DIRECT',FORM='UNFORMATTED',
     2    RECL=1)
      IF (IOS .GT. 0 ) THEN deal with error  ENDIF

      VBS = 4
      VN  = 2
      HBS = 3
      HN  = 3
      SUBFILE = 5
      FLAGS = IOR (ACTIVE, NORMAL)
      RT = PIOFSCHANGEVIEW(7, VBS, VN, HBS, HN, SUBFILE, FLAGS)

C CHECK THE RETURNED ERROR VALUE
       IF (RT .NE. 0) THEN deal with error  ENDIF
```

piofs_chkpoint

Purpose

piofs_chkpoint – Establish a checkpoint for a Parallel I/O File System file.

C Syntax

```
#include <piofs/piofs_ioctl.h>

int piofsioctl(
   int fildes,
   int PIOFS_CHKPOINT,
   void *argument );
```

Parameters

fildes Specifies a previously opened file descriptor returned by the AIX **open()** or **creat()** function.

PIOFS_CHKPOINT
Specifies the **piofsioctl** operation to be performed on the device: Take a checkpoint of the open Parallel I/O File System file.

argument A NULL pointer.

FORTRAN Syntax

```
INTEGER FUNCTION PIOFSCHKPT (unit)

INTEGER*4  unit
```

unit Unit number of the file

Description

This function takes a checkpoint of the open file, replacing any existing checkpoint of the file. Strictly speaking the file descriptor supplied with the **piofsioctl()** function call refers to a subfile of a Parallel I/O File System file and not the entire file. However, for PIOFS_CHKPOINT the entire file is checkpointed not just the subfile referred to by the file descriptor. The function is synchronous. After completion, the checkpointed version of the file can be copied to another file. The checkpointed file can also be restored to active file status by calling “piofs_restore_chkpoint” on page 118.

PIOFS_CHKPOINT requires no explicit coordination with other tasks in the job. However, the most reasonable way to use it is to take a checkpoint when there is no concurrent activity that may modify the file, for example, no writes. If the file is being written by any node concurrently with the checkpointing, then the contents of the file at checkpoint time may be unpredictable.

If application level checkpointing is implemented, then PIOFS_CHKPOINT can be used as a component of the application checkpointing algorithm. The checkpoint of a file is discarded when that file is deleted.

If the checkpoint fails, any previous checkpoint of the file will be deleted.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EBADF The file descriptor *fildes* or *unit* is not a valid open file descriptor or unit number.

EIO Internal communication error.

Examples

1. C Example.

   To checkpoint *thisfile*:

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>

int   rt,
      fd;

fd = open('/piofs/gooduser/thisfile',O_RDWR, 0740);

/*************************************/
/*    do other things                */
/*************************************/

/* checkpoint the file */
rt = piofsioctl(fd,PIOFS_CHKPOINT, NULL);

/*************************************/
/*     check for errors              */
/*************************************/
```

2. FORTRAN example.

   To checkpoint *thisfile*:

```
       INCLUDE 'piofs/piofs.inc'

       INTEGER*4  RT, IOS

C Open the file as unit=7 and do other things
       OPEN(UNIT=7, IOSTAT=IOS, FILE='/piofs/gooduser/thisfile',
      1   STATUS='OLD', ACCESS='DIRECT', FORM='UNFORMATTED',
      2   RECL=1)
C
C  do other things
C

       RT = PIOFSCHKPT( 7 )


C  CHECK THE RETURNED ERROR VALUE
       IF (RT .NE. 0) THEN deal with error  ENDIF
```

piofs_create

Purpose

piofs_create – Create a Parallel I/O File System file with a specified number of cells and a specified basic striping unit.

C Syntax

#include <**piofs**/**piofs.h**>

int piofsioctl(
int *fildes*,
int PIOFS_CREATE,
piofs_creat_t **argument* **);**

Parameters

fildes Is ignored and should be 0.

PIOFS_CREATE
Specifies the **piofsioctl** operation to be performed on the device. Create a Parallel I/O File System file.

argument A pointer to a **piofs_create_t** structure:

```
typedef struct {
  char name[PIOFS_PATH_MAX + 1]; /* Fully qualified file name*/
  int bsu;                       /* Basic striping unit in bytes */
  int cells;                     /* Number of cells              */
  int permissions                /* Access permissions           */
  int base_node                  /* Node to hold the first cell  */
  int flags;                     /* 0 or REPLACE                 */
} piofs_create_t ;
```

The flags field values are as follows:

0 Do not replace an existing file with the same name.

REPLACE Replace an existing file of the same name. If REPLACE is not specified, an attempt to overwrite an existing file will be unsuccessful and will return an error.

Permissions can be specified either as octal values or, if you also include the **sys**/**mode.h** header file, as the typical `-rwxrwxrwx` values.

FORTRAN Syntax

INTEGER FUNCTION PIOFSCREATE (*name*, *bsu*, *cells*, *permissions*, *base_node*, *flags***)**

INTEGER*4 cells, bsu, permissions, base_node, flags
CHARACTER*255 name

name The fully qualified file name

bsu Basic striping unit in bytes. Specify -1 to use the default value of 32,768 bytes.

cells Number of cells that will be created. Specify -1 to use the default value of one cell per node.

permissions AIX file permissions, usually given as an octal number

base_node The specific node to be used as the base node for this file. The first cell will be placed on the base_node. Specify -1 to use the recommended choice and the default — pseudo-random base node selection.

flags One of the following:

0 Do not replace an existing file of the same name as *name*

REPLACE Replace an existing file of the same name. If REPLACE is not specified, an attempt to overwrite an existing file will be unsuccessful and will return an error.

Description

This function causes PIOFS to create a file with the specified file name and the specified basic striping unit and number of cells. If the function call results in the creation of a file, the file is created with 0 bytes. There are no defaults for these parameters. All parameters must be valid.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EACCES One of:

- Permission denied
- The file is being used by another user.

EEXIST A file with the same name already exists. This will not occur if the REPLACE flag is set.

EINVAL One of the following is true:

- The permissions are not valid file creation permissions.
- The *flag* is neither 0 nor REPLACE.
- The *bsu* parameter is invalid.
- The *name* is empty, too long, or otherwise invalid.

EMFILE No space for additional objects in system tables.

EPERM Permission denied, not owner.

Examples

1. C Example:

   To create a file with a basic striping unit of 1024 and with 8 cells:

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>
#include <string.h>

int          rt,
             fd;
piofs_create_t mycreat;

strcpy(mycreat.name,"/piofs/thisuser/newfile");
mycreat.bsu        = 1024;
mycreat.cells      = 8;
mycreat.permissions= 0660;
mycreat.flags      = REPLACE;
mycreat.base_node  = -1;


rt = piofsioctl(fd,PIOFS_CREATE, &mycreat);
if (rt != 0) /* look at errno for more details */ ;
```

2. FORTRAN Example:

   To create a file with a basic striping unit (bsu) of 1024 and with 8 cells:

```
       INCLUDE 'piofs/piofs.inc'

       INTEGER*4       BSU, CELLS, PERMISSIONS, FLAGS, BASE_NODE
       CHARACTER*128   FN
       INTEGER*4       RT

       FN  = '/piofs/thisuser/newfile'
       BSU               = 1024
       CELLS             = 8
       PERMISSIONS       = O'660'
       FLAGS             = REPLACE
       BASE_NODE         = -1
C
       RT = PIOFSCREATE(FN, BSU, CELLS, PERMISSIONS, BASE_NODE, FLAGS)
C  check for an error
       IF (RT .NE.  0)  THEN deal with error ENDIF
```

piofs_delete_chkpoint

Purpose

piofs_delete_chkpoint – Delete the checkpointed version of a Parallel I/O File System file.

C Syntax

#include <**piofs/piofs_ioctl.h**>

int piofsioctl(
int *fildes*,
int PIOFS_DELETE_CHKPOINT,
void **argument* **);**

Parameters

fildes Specifies a previously opened file descriptor returned by the AIX **open()** or **creat()** function.

PIOFS_DELETE_CHKPOINT
Specifies the **piofsioctl** operation to be performed on the device: Delete the checkpointed version of an open Parallel I/O File System file.

argument A NULL pointer.

FORTRAN Syntax

INTEGER FUNCTION PIOFSDELCHKPT (*unit***)**

INTEGER*4 unit

unit Unit number of the file

Description

This function deletes the checkpointed version of a Parallel I/O File System file. All disk space used only by the checkpointed version of the file is freed. If the file had no checkpointed version, an error is returned.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EBADF The file descriptor *fildes* or *unit* is not a valid open file descriptor or unit number.

EIO Internal communication error.

ENOENT There is no checkpointed version of the file to remove.

Examples

1. C Example.

   To delete the current checkpointed version of a file:

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>

int fd, rt ;

fd = open('/piofs/gooduser/thisfile',O_RDWR,0740);

rt = piofs_ioctl(fd,PIOFS_DELETE_CHKPOINT, NULL);
```

2. FORTRAN example.

   To delete the current checkpointed version of a PIOFS file:

```
       INCLUDE    'piofs/piofs.inc'

       INTEGER*4   RT, IOS

C Open the file as unit=7 and do other things
       OPEN(UNIT=7, IOSTAT=IOS, FILE='/piofs/gooduser/thisfile',
      1   STATUS='OLD', ACCESS='DIRECT', FORM='UNFORMATTED',
      2   RECL=1)
C
C  do other things
C

       RT = PIOFSDELCHKPT( 7 )

       IF (RT .NE. 0 ) THEN deal with error  ENDIF
```

piofs_fstat

Purpose

piofs_fstat — Retrieve the status of a Parallel I/O File System file.

C Syntax

#include <**piofs/piofs_ioctl.h**>

int piofsioctl(
int *fildes*,
int PIOFS_FSTAT,
piofs_fstat_t **argument* **);**

Parameters

fildes Specifies a previously opened file descriptor returned by the AIX **open()** function.

PIOFS_FSTAT
Specifies the **piofsioctl** operation to be performed on the device: Retrieve the status of a Parallel I/O File System file.

argument A pointer to a piofs_fstat_t structure:

```
typedef struct {
   ushort_t st_mode;           /* type of file and permissions    */
   uid_t    st_uid;            /* uid of file owner               */
   gid_t    st_gid;            /* gid of file owner               */
                               /* Important: the definition of
                                  the variable type, offset_t, differs
                                  between PIOFS for AIX Version 3.2.5
                                  and PIOFS for AIX Version 4.1.
                                  Check its definition in
                                  piofs/piofs.h                   */
   offset_t st_size;           /* size of active file             */
   offset_t st_chkpt_size;     /* size of checkpoint              */
   time_t   st_atime;          /* last access time                */
   time_t   st_mtime;          /* last data modification time     */
   time_t   st_ctime;          /* create time                     */
   ulong_t  st_blksize;        /* optimal fs blocksize for I/O    */
   offset_t st_blocks;         /* number of DEV_BSIZE blocks      */
   int      st_vfstype;        /* type of fs                      */
   ulong_t  st_vfs;            /* vfs number                      */
   ulong_t  st_type;           /* vnode type                      */
   int      st_checkpoint;     /* TRUE if checkpoint exists       */
   int      st_cells;          /* number of cells                 */
   int      st_bsu;            /* basic striping unit             */
   int      st_base_node;      /* node at which first cell resides*/
   int      st_flags;          /* flags from piofs_change_view:
                                 ACTIVE/CHECKPOINT/NORMAL/CAUTIOUS */
   ulong_t  reserved[2];       /* reserved by the system           */
} piofs_fstat_t;
```

FORTRAN Syntax

INTEGER FUNCTION PIOFSFSTAT (*unit*, *mode*, *uid*, *gid*, *size*, *chkpt_size*, *atime*, *mtime*, *ctime*, *blksize*, *blocks*, *vfstype*, *vfs*, *type*, *chkpoint*, *cells*, *bsu*, *base_node*, *flags* **)**

INTEGER*2 mode
INTEGER*4 unit, uid, gid, atime, mtime, ctime
INTEGER*4 blksize, vfstype, vfs, type, cells, bsu, base_node, flags
INTEGER*8 size, chkpt_size, blocks
LOGICAL chkpoint

unit The unit number of the file

mode The file type and permission bits; see **sys/mode.h**

uid The userid of the file owner

gid The group id of the file owner

size The size of the active file

chkpt_size The size of the checkpoint version of the file

atime Last access time

mtime Last data modification time

ctime File creation time

blksize Optimal fs block size for I/O

blocks Number of DEV_BSIZE blocks; see **sys/dir.h**

vfstype Type of file system; see **vnode.h**

vfs Vfs number

type Vnode type

chkpoint

.TRUE. If the file has a checkpoint

.FALSE. If the file does not have a checkpoint

cells The number of cells allocated for this file

bsu The basic striping unit in bytes

base_node The node at which the first cell resides

flags Flags from piofs_change_view:
ACTIVE/CHECKPOINT/NORMAL/CAUTIOUS

Description

This function returns status information about Parallel I/O File System files.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EBADF	The file descriptor *fildes* or *unit* is not a valid open file descriptor or unit number.
EIO	Internal communication error.

Examples

1. C example

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>

int     rt;
piofs_fstat_t buf;

fd = open('/piofs/gooduser/thisfile',O_RDWR, 0740);

/*************************************/
/*    do other things                */
/*************************************/

/* get the status of the file */

rt = piofsioctl(fd, PIOFS_FSTAT, &buf);
```

2. FORTRAN example

```
       INCLUDE 'piofs/piofs.inc'

       INTEGER*2  MODE
       INTEGER*4  CREATE_TIME, OWNER, TYPE, RT, IOS
       INTEGER*4  CELLS, GID, MTIME, ATIME, BSU, BASE_NODE, FLAGS
       INTEGER*4  BLKSIZE, VFSTYPE, VFS, TYPE
       INTEGER*8  FILE_SIZE, CHKPT_SIZE, BLOCKS
       LOGICAL    CHKPOINT

C Open the file as unit=7
       OPEN(UNIT=7, IOSTAT=IOS, FILE='/piofs/gooduser/thisfile',
      1   STATUS='OLD', ACCESS='DIRECT', FORM='UNFORMATTED',
      2   RECL=1)
C
       RT = PIOFSFSTAT( UNIT, MODE, OWNER, GID, FILE_SIZE,
      X    CHKPT_SIZE, ATIME, MTIME, CREATE_TIME,
      X    BLKSIZE, BLOCKS, VFSTYPE, VFS, TYPE,
      X    CHKPOINT, CELLS, BSU, BASE_NODE, FLAGS)

C  check THE RETURNED ERROR VALUE
       IF (RT .NE. 0 ) THEN deal with error  ENDIF
```

piofs_lseek

Purpose

piofs_lseek – Sets or determines the Parallel I/O File System subfile offset value. It supports a file offset of 2 Gigabytes or greater.

C Syntax

```
#include <piofs/piofs_ioctl.h>
#include <unistd.h>

int piofsioctl(
   int fildes,
   int PIOFS_LSEEK,
   piofs_lseek_t *argument );
```

Parameters

fildes Specifies the a previously opened file descriptor returned by the AIX **open()** or **creat()** function.

PIOFS_LSEEK
Specifies the **piofsioctl** operation to be performed: set or determine the offset for a Parallel I/O File System file.

argument Points to a **piofs_lseek** structure:

```
typedef struct {
                        /* IMPORTANT: The definition of
                           the variable type, offset_t, differs
                           between PIOFS for AIX Version 3.2.5 and
                           PIOFS for AIX Version 4.1. Be sure to check
                           its definition piofs/piofs.h.  */
  offset_t offset;    /* 64-bit file offset in bytes             */
  int      whence;    /* SEEK_SET, SEEK_CUR, or SEEK_END         */
  offset_t  setto;    /* file offset when PIOFS_LSEEK completes  */
} piofs_lseek_t ;
```

The field values are as follows:

whence

SEEK_SET	The offset is set to *offset* bytes.
SEEK_CUR	The offset is set to its current value plus *offset* bytes.
SEEK_END	The offset is set to the size of the file plus *offset* bytes. See "Description" on page 116 for details.

setto The 64-bit offset that the file pointer is at after **piofs_lseek** has completed.

Description

This function causes the Parallel I/O File System to use its own offset and to not use the AIX file offset. Another way to use the Parallel I/O File System offset instead of the AIX file offset is to add the following line to a **piofs.profile** file:

```
USE_PIOFS_LSEEK
```

See "Accessing PIOFS Files Correctly" on page 92 for more information.

You must have set the file offset with **PIOFS_LSEEK** before changing the offset with *whence* set to SEEK_CUR or SEEK_END.

SEEK_END can be used only with the view set to *Vn* = *Vbs* = *Hn* = *Hbs* = 1 (normal sequential access to the file).

Once the PIOFS_LSEEK **piofsioctl** command is issued, the standard **lseek()** function will return normally but will have no effect on the offset. The file offset will be changed appropriately only as the result of a **read()** or **write()**, or by the **PIOFS_LSEEK** function.

SEEK_SET, SEEK_CUR, and SEEK_END are defined in the **/usr/include/unistd.h** header file.

To determine the current offset of a Parallel I/O File System file, use **piofs_lseek** with *whence* set to SEEK_CUR and *offset* set to 0.

Note that if you use **fread** and **fwrite** and also wish to use a **piofsioctl** option such as PIOFS_LSEEK, you must obtain the file descriptor corresponding to the file pointer used for **fread** or **fwrite**. This can be done using the AIX **fileno()** macro. Refer to the **fileno** man page for details.

Return Values

This function returns zero if it completes successfully. If an error occurs, the function will return –1 and the error code is set in the variable **errno**. The following non-zero values are set to indicate an error.

EBADF The file descriptor *fildes* or *unit* is not a valid open file descriptor or unit number.

EINVAL The *whence* parameter is not a proper value or the resulting file offset would be invalid.

Examples

C Example

To begin writing at offset 2 Gigabytes in subfile 0 of a file.

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>
#include <unistd.h>

int   rt, fd;
piofs_lseek_t myseek;
/* This is example uses the PIOFS AIX Version 3.2.5 definition for offset_t */
myseek.offset.h = 0;
myseek.offset.l = 0x80000000;
myseek.whence   = SEEK_SET;

/* The default is to open one subfile, that subfile */
/* is subfile zero.                                 */
fd = open("/piofs/thisuser/myfile",O_WRONLY, 0740);

rt = piofsioctl(fd,PIOFS_LSEEK, &myseek);
```

piofs_restore_chkpoint

Purpose

piofs_restore_chkpoint – Restore a checkpoint of a Parallel I/O File System file.

C Syntax

#include <**piofs/piofs_ioctl.h**>

int piofsioctl(
int *fildes***,**
int PIOFS_RESTORE_CHKPOINT,
void **argument* **);**

Parameters

fildes Specifies a previously opened file descriptor returned by the AIX **open()** or **creat()** function.

PIOFS_RESTORE_CHKPOINT
Specifies the **piofsioctl** operation to be performed on the device: Restore the checkpoint of a file.

argument A NULL pointer.

FORTRAN Syntax

INTEGER FUNCTION PIOFSRCHKPT (*unit***)**

INTEGER*4 unit

unit Unit number of the file

Description

This function restores a checkpoint of a Parallel I/O File System file, replacing the existing active version of the file. The function is synchronous; it blocks the calling task. After completion, the contents of the active version of the file will be identical to the contents of the file when it was last checkpointed.

PIOFS_RESTORE_CHKPOINT requires no explicit coordination with other tasks in the job. However, the only reasonable way to use it is to restore the file when there is no concurrent activity involving the file. If the file is being modified from any node concurrently with the restoration, then the contents of the restored file are *unpredictable*. If the file is being read concurrently with the restoration, these accesses may return either the active version, the newly restored one, or a mix of the two.

After the checkpoint is restored, it nevertheless stays in effect. It is possible to use PIOFS_DELETE_CHECKPOINT to delete the checkpoint and free its disk space.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EBADF The file descriptor *fildes* or *unit* is not a valid open file descriptor or unit number.

EIO Internal communication error.

255 Checkpoint of file does not exist. (There is no standard error code for this event.)

Examples

1. C example

   To restore a checkpoint:

```
#include <piofs/piofs.h>
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>

int   rt, fd;

rt = piofsioctl(fd,PIOFS_RESTORE_CHKPOINT, NULL );
```

2. FORTRAN example

   To restore a checkpoint:

```
       INCLUDE 'piofs/piofs.inc'

       INTEGER*4   RT

C
C Unit 7 is connected to an opened Parallel I/O File System file.
C

       CALL flush_(7)
       RT = PIOFSRCHKPT(7 )

       IF (RT .NE. 0) THEN deal with error  ENDIF
```

piofs_statfs

Purpose

piofs_statfs – Return system status information about all server nodes.

C Syntax

#include <**piofs/piofs_ioctl.h**>

int piofsioctl(
int fildes,
int PIOFS_STATFS,
piofs_statfs_t ******argument* **);**

Parameters

fildes This parameter is ignored. It should be zero.

PIOFS_STATFS
Specifies the **piofsioctl** operation to be performed: Return system status information about all server nodes.

argument A pointer to a **piofs_statfs_t** structure:

```
typedef struct
  { char   name[PIOFS_PATH_MAX+1]; /* name of the mount point */
    int      f_nodes;            /* number of server nodes  */
    int      f_bsize;            /* file system block size   */
                                 /* Important: the definition of
                                    the variable type, offset_t,
                                    differs between PIOFS for
                                    AIX Version 3.2.5 and PIOFS for
                                    AIX Version 4.1. Check its
                                    definition in
                                    piofs/piofs.h                */
    offset_t f_blocks;           /* total data blocks       */
    offset_t f_bavail;           /* available data blocks   */
    offset_t f_files;            /* total objects possible  */
    offset_t f_ffree;            /* free objects            */
    ulong_t  reserved[4];        /* reserved                */
  } piofs_statfs_t;
```

FORTRAN Syntax

INTEGER FUNCTION PIOFSSTATFS (*name*, *f_nodes*, *f_bsize*, *f_blocks*, *f_bavail*, *f_files*, *f_ffree***)**

```
INTEGER*4      f_nodes, f_bsize
INTEGER*8      f_blocks, f_bavail, f_files, f_ffree
CHARACTER*256  name
```

name Name of the mount point.

f_nodes Number of server nodes.

f_bsize File system block size.

f_blocks Total data blocks.

f_bavail Number of available data blocks.

f_files Total number of objects possible.

f_ffree Number of free objects.

Description

The PIOFS_STATFS command returns status information about the entire Parallel I/O File System. You supply the name of the mount point; the function returns the information in the other fields of the **piofs_statfs_t** structure.

Return Values

Both the C and FORTRAN functions return zero if they complete successfully. If an error occurs the C function will return –1 and the error code is set in the variable **errno**. For the FORTRAN function the error code is set in the returned value. For both functions the following non-zero values are set to indicate an error.

EIO Internal communication error.

ENOMEM No space for additional objects in system tables.

EINVAL The **name** field is not a valid mount point for a Parallel I/O File System.

Examples

1. C Example

   To get status information about the Parallel I/O File System:

```
#include <piofs/piofs_ioctl.h>
#include <fcntl.h>
#include <strings.h>

int     rt;
piofs_statfs_t buf;

strcpy(buf.name, "/piofs");
rt = piofsioctl(fd, PIOFS_STATFS,&buf);
```

2. FORTRAN example

   To get status information about the Parallel I/O File System:

piofs_statfs

```
      INCLUDE 'piofs/piofs.inc'

      CHARACTER*256  NAME
      INTEGER*4      F_NODES, F_BSIZE, RT
      INTEGER*8      F_BLOCKS, F_BAVAIL, F_FILES, F_FFREE

      NAME = '/piofs'
C
      RT = PIOFSSTATFS( NAME, F_NODES, F_BSIZE, F_BLOCKS,
     X    F_BAVAIL, F_FILES, F_FFREE)

C  check THE RETURNED VALUE
      IF (RT .NE. 0 ) THEN deal with error  ENDIF
```

AIX Subroutines with Parallel I/O File System Considerations

AIX Subroutine Documentation Conventions. 124
Error Codes for Services that Require Path Name Resolution 125
access, accessx, or faccessx 126
chmod or fchmod 129
chown, fchown, chownx, or fchownx 133
close 137
fclose or fflush 139
fcntl, dup2, or dup 141
fopen, freopen, or fdopen 149
fread or fwrite 154
fseek, rewind, ftell, fgetpos, or fsetpos 157
getc, getchar, fgetc, or getw 161
llseek (AIX Version 4.1) 164
lseek 166
mkdir 168
open, openx, creat 171
putc, putchar, fputc, or putw 179
puts or fputs 182
read, readx, readv, or readvx 184
remove 190
rename 191
rmdir 194
stat, lstat, fstat, statx, fstatx, fullstat, or ffullstat 197
unlink 202
write, writex, writev, or writevx 204

© Copyright IBM Corp. 1995

AIX Subroutine Documentation Conventions.

This section contains reference pages for AIX subroutines that have changes when used with the Parallel I/O File System. The considerations are marked using brackets as follows:

Not Applicable to PIOFS

The Parallel I/O File System does not support this action.

End of Not Applicable to PIOFS

Applicable only to PIOFS

The Parallel I/O File System adds this action to the subroutine.

End of Applicable only to PIOFS

Error Codes for Services that Require Path Name Resolution

The following errors apply to any service that requires path name resolution.

EACCES Search permission is denied on a component of the path prefix.

EFAULT The *Path* parameter points outside of the allocated address space of the process.

EIO An I/O error occurred during the operation.

Not Applicable to PIOFS

ELOOP Too many symbolic links were encountered in translating the *Path* parameter.

ENAMETOOLONG A component of a path name exceeded 255 characters and the process has the *DisallowTruncation* attribute (see the **ulimit** subroutine), or an entire path name exceeded 1023 characters.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG A component of a path name exceeded 255 characters or an entire path name exceeded 255 characters.

End of Applicable only to PIOFS

ENOENT A component of the path prefix does not exist.

Not Applicable to PIOFS

ENOENT A symbolic link was named, but the file to which it refers does not exist.

End of Not Applicable to PIOFS

ENOENT The path name is null.

ENOTDIR A component of the path prefix is not a directory.

ESTALE The root or current directory of the process is located in a virtual file system that is unmounted.

access, accessx, or faccessx

Purpose

access, accessx, faccessx – Determines the accessibility of a file

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <unistd.h>
#include <sys/access.h>

int access(PathName, Mode)
char *PathName;
int Mode;

int accessx(PathName, Mode, Who)
char *PathName;
int Mode;
int Who;

int faccessx(FileDescriptor, Mode, Who)
int FileDescriptor;
int Mode;
int Who;
```

Description

The **access**, **accessx**, and **faccessx** subroutines determine the accessibility of a file system object. The **accessx**, and **faccessx** subroutines allow the specification of a class of users or processes for whom access is to be checked.

Parameters

PathName Specifies the pathname of the file. If the *PathName* parameter refers to a symbolic link, the **access** subroutine returns information about the file pointed to by the symbolic link.

FileDescriptor
Specifies the FileDescriptor of an open file.

Mode Specifies the access modes to be checked. This parameter is a bitmask containing zero or more of the following values, which are defined in the **sys/mode.h** file:

R_ACC	Check read permission
W_ACC	Check write permission
X_ACC	Check execute or search permission
E_ACC	Check the existence of a file

If none of these values is specified, the existence of a file is checked.

who Specifies the class of users for whom access is to be checked. This parameter must be one of the following values, defined in the **sys/access.h** header file:

ACC_SELF Determines if access is permitted for the current process. The effective user and group IDs, the concurrent group set, and the privilege of the current process are used for the calculation.

ACC_INVOKER Determines if access is permitted for the invoker of the current process. The real user and group IDs, the concurrent group set and the privilege of the invoker are used for the calculation.

Note: The expression **access**(*pathname, mode*) is equivalent to **accessx**(*pathname, mode*, **ACC_INVOKER**).

ACC_OTHERS Determines if the specified access is permitted for any user other than the object owner. The *Mode* parameter must contain only one of the valid modes. Privilege is not considered in the calculation.

ACC_ALL Determines if the specified access is permitted for all users. The *Mode* parameter must contain only one of the valid modes. Privilege is not considered in the calculation.

Return Values

If the requested access is permitted, the **access**, **accessx**, and **faccessx** subroutines return 0. If the requested access is not permitted or the function call fails, –1 is returned and **errno** is set to indicate the error.

Error Codes

The **access** and **accessx** subroutines fail if one or more of the following are true:

EACCES Search permission is denied on a component of the *PathName* prefix.

EFAULT The *pathname* parameter points to a location outside the allocated address space of the process.

Not Applicable to PIOFS

ELOOP Too many symbolic links were encountered in translating the *PathName* parameter.

End of Not Applicable to PIOFS

ENOENT A component of the *pathname* does not exist or the process has the disallow truncation attribute set.

ENOTDIR A component of the *PathName* is not a directory.

ENOENT The named file does not exist.

ENOENT The *PathName* parameter was null.

Not Applicable to PIOFS

ENOENT A symbolic link was named, but the file to which it refers does not exist.

ENAMETOOLONG A component of a path name exceeded 255 characters and the process has the *DisallowTruncation* attribute (see the **ulimit** subroutine), or an entire path name exceeded 1023 characters.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG A component of a path name exceeded 255 characters or an entire path name exceeded 255 characters.

End of Applicable only to PIOFS

ESTALE The process's root or current directory is located in a virtual file system that has been unmounted.

The **faccessx** subroutine fails if the following is true:

EBADF The value of the *FileDescriptor* parameter is not valid.

The **access**, **accessx**, and **faccessx** subroutines fail if one or more of the following is true:

EIO An I/O error occurred during the operation.

EACCES The file protection does not allow the requested access.

EROFS Write access is requested for a file on a read-only file system.

Not Applicable to PIOFS

If NFS is installed on your system, the **accessx** and **faccessx** subroutines can also fail if the following is true:

End of Not Applicable to PIOFS

ETIMEOUT The connection timed out.

Related Information

Commands: **chmod**, **chown**.

chmod or fchmod

Purpose

chmod or **fchmod** – Changes file access permissions.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <sys/stat.h>

int chmod (Path, Mode)
const char *Path;
mode_t Mode;

int chmod (FileDescriptor, Mode)
int FileDescriptor;
mode_t Mode;
```

Description

The **chmod** subroutine sets the access permissions of the file specified by the *Path* parameter. If Network File System (NFS) is installed on your system, this path can cross into another node.

Use the **fchmod** subroutine to set the access permissions of an open file pointed to by the *FileDescriptor* parameter.

The access control information is set according to the *Mode* parameter.

Parameters

FileDescriptor Specifies the file descriptor of an open file.

Mode Specifies the bit pattern that determines the access permissions. The *Mode* parameter is constructed by logically ORing one or more of the following values, which are defined in the **sys/mode.h** file:

S_ISUID Enables the **setuid** attribute for an executable file. A process executing this program acquires the access rights of the owner of the file.

S_ISGID Enables the **setgid** attribute for an executable file. A process executing this program acquires the access rights of the group of the file. Also, enables the group-inheritance attribute for a directory. Files created in this directory have a group equal to the group of the directory.

The following attributes apply only to files that are directly executable. They have no meaning when applied to executable text files such as shell scripts and awk scripts.

Not Applicable to PIOFS

S_ISVTX Enables the **link**/**unlink** attribute for a directory. Files cannot be linked to in this directory. Files can only be unlinked if the requesting process has write permission for the directory and is either the owner of the file or the directory.

S_ISVTX Enables the save text attribute for an executable file. The program is not unmapped after usage.

S_ENFMT Enables enforcement-mode record locking for a regular file. File locks requested with the **lockf** subroutine are enforced.

End of Not Applicable to PIOFS

S_IRUSR Permits the file's owner to read it.

S_IWUSR Permits the file's owner to write to it.

S_IXUSR Permits the file's owner to execute it (or to search the directory).

S_IRGRP Permits the file's group to read it.

S_IWGRP Permits the file's group to write to it.

S_IXGRP Permits the file's group to execute it (or to search the directory).

S_IROTH Permits others to read the file.

S_IWOTH Permits others to write to the file.

S_IXOTH Permits others to execute the file (or to search the directory).

Other mode values exist that can be set with the **mknod** subroutine but not with the **chmod** subroutine.

Path Specifies the full path name of the file.

Return Values

Upon successful completion, the **chmod** subroutine and **fchmod** subroutines return a value of 0. If the **chmod** subroutine or **fchmod** subroutine is unsuccessful, a value of -1 is returned, and the **errno** global variable is set to identify the error.

Error Codes

The **chmod** subroutine is unsuccessful and the file permissions remain unchanged if one of the following is true:

ENOTDIR A component of the *Path* prefix is not a directory.

EACCES Search permission is denied on a component of the *Path* prefix.

EFAULT The *Path* parameter points to a location outside of the allocated address space of the process.

ESTALE The root or current directory of the process is located in a virtual file system that has been unmounted.

Not Applicable to PIOFS

ELOOP Too many symbolic links were encountered in translating the *Path* parameter.

End of Not Applicable to PIOFS

ENOENT A symbolic link was named, but the file to which it refers does not exist.

ENOENT A component of the *Path* parameter does not exist or has the disallow truncation attribute (see the **ulimit** subroutine).

ENOENT The *Path* parameter is NULL.

ENOENT The named file does not exist.

Not Applicable to PIOFS

ENAMETOOLONG A component of a path name exceeded 255 characters and the process has the *DisallowTruncation* attribute (see the **ulimit** subroutine), or an entire path name exceeded 1023 characters.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG A component of a path name exceeded 255 characters or an entire path name exceeded 255 characters.

End of Applicable only to PIOFS

EPERM The effective user ID does not match the owner of the file, and the process does not have appropriate privileges.

EINVAL The value of the *Mode* parameter is invalid.

The **fchmod** subroutine is unsuccessful and the file permissions remain unchanged if the following is true:

EBADF The value of the *FileDescriptor* parameter is not valid.

The **chmod** or **fchmod** subroutine is unsuccessful and the access control information for a file remains unchanged if one of the following is true:

EROFS The named file resides on a read-only file system.

EIO An I/O error occurred during the operation.

Not Applicable to PIOFS

EBUSY The value of the Mode parameter would change the enforced locking attribute of an open file.

If NFS is installed on your system, the **chmod** and **fchmod** subroutines can also be unsuccessful if the following is true:

End of Not Applicable to PIOFS

ETIMEDOUT The connection timed out.

Security

Access Control The invoker must have search permission for all components of the *Path* prefix.

Related Information

Commands: **aclget, aclput, chmod**

Not Applicable to PIOFS

Subroutines: **acl_chg, acl_get, acl_put, acl_set, chacl, statacl, stat**.

End of Not Applicable to PIOFS

chown, fchown, chownx, or fchownx

Purpose

chown – Changes file ownership.

C Syntax

Syntax for the **chown** and **fchown** subroutines:

```
#include <sys/types.h>

int chown (Path, Owner, Group)
const char *Path;
uid_t Owner;
gid_t Group;

int chown (FileDescriptor, Owner, Group)
in FileDescriptor;
uid_t Owner;
gid_t Group;
```

Syntax for the **chownx** and **fchownx** subroutines:

```
#include <sys/types.h>
#include <sys/chownx.h>

int chownix (Path, Owner, Group, Flags)
const char *Path;
uid_t Owner;
gid_t Group;
int Flags;

int fchownix (FileDescriptor, Owner, Group, Flags)
in FileDescriptor;
uid_t Owner;
gid_t Group;
int Flags;
```

Description

The **chown, chownx, fchown,** and **fchownx** subroutines set the file owner and group IDs of the specified file system object. Root user authority is required to change the owner of a file.

Parameters

FileDescriptor Specifies the file descriptor of an open file.

Flags Specifies whether the file owner ID or group ID should be changed. This parameter is constructed by logically ORing the following values:

T_OWNER_AS_IS Ignores the value specified by the *Owner* parameter and leaves the owner ID of the file unaltered.

T_GROUP_AS_IS Ignores the value specified by the *Group* parameter and leaves the group ID of the file unaltered.

Group Specifies the new group of the file. If this value is -1, the group is not changed. (A value of -1 indicates only that the group is not changed; it does not indicate a group that is not valid. An owner or group ID cannot be invalid.)

Owner Specifies the new owner of the file. If this value is -1, the owner is not changed. (A value of -1 indicates only that the group is not changed; it does not indicate a group that is not valid. An owner or group ID cannot be invalid.)

Path Specifies the full path name of the file. If *Path* resolves to a symbolic link, the ownership of the file or directory pointed to by the symbolic link is changed.

Return Values

Upon successful completion, the **chown, chownx, fchown,** and **fchownx** subroutines return a value of 0. If the **chown, chownx, fchown,** or **fchownx** subroutine is unsuccessful, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **chown** or **chownx** subroutine is unsuccessful and the owner and group of a file remain unchanged if one of the following is true:

EACCESS Search permission is denied on a component of the *Path* parameter.

Not Applicable to PIOFS

EDQUOT The new group for the file system object cannot be set because the group's quota of disk blocks or i-nodes has been exhausted on the file system.

End of Not Applicable to PIOFS

EFAULT The *Path* parameter points to a location outside of the allocated address space of the process.

EINVAL The owner or group ID supplied is not valid.

Not Applicable to PIOFS

ELOOP Too many symbolic links were encountered in translating the *Path* parameter.

End of Not Applicable to PIOFS

Not Applicable to PIOFS

ENAMETOOLONG A component of the *Path* parameter exceeded 255 characters, or the entire *Path* parameter exceeded 1023 characters.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG A component of the *Path* parameter exceeded 255 characters or the entire *Path* parameter exceeded 255 characters.

End of Applicable only to PIOFS

ENOENT A symbolic link was named, but the file to which it refers does not exist; or a component of the *Path* parameter does not exist; or the process has the disallow truncation attribute set; or the *Path* parameter is NULL.

ENOTDIR A component of the path prefix is not a directory.

EPERM The effective user ID does not match the owner of the file, and the calling process does not have the appropriate privileges.

Not Applicable to PIOFS

EROFS The named file resides on a read-only file system.

End of Not Applicable to PIOFS

ESTALE The root or current directory of the process is located in a virtual file system that has been unmounted.

The **fchown** or **fchownx** subroutine is unsuccessful and the file owner and group remain unchanged if one of the following is true:

EBADF The named file resides on a read-only file system.

Not Applicable to PIOFS

EDQUOT The new group for the file system object cannot be set because group's quota of disk blocks or i-nodes has been exhausted on the file system.

End of Not Applicable to PIOFS

EIO An I/O error occurred during the operation.

Security

Access Control The invoker must have search permission for all components of the *Path* parameter.

Auditing Events

Event FILE_SetOwner

Information	Object descriptor, owner, group.

Related Information

The **chmod** subroutine.

close

Purpose

close – Closes the file associated with a file descriptor.

C Syntax

#include <**unistd.h**>

int close (*FileDescriptor***)**
int *FileDescriptor***;**

Description

The **close** subroutine closes the file associated with the *FileDescriptor* parameter. If Network File System (NFS) is installed on your system, this file can reside on another node.

Not Applicable to PIOFS

All file regions associated with the file specified by the *FileDescriptor* parameter that this process has previously locked with the **lockf** or **fcntl** subroutine are unlocked. This occurs even if the process still has the file open by another file descriptor.

If the *FileDescriptor* parameter resulted from an **open** subroutine that specified **O_DEFER**, and this was the last file descriptor, all changes made to the file since the last **fsync** subroutine are discarded.

If the *FileDescriptor* parameter is associated with a mapped file, it is unmapped.The shmat subroutine provides more information about mapped files.

End of Not Applicable to PIOFS

The **close** subroutine attempts to cancel outstanding asynchronous I/O requests on this file descriptor. If the asynchronous I/O requests cannot be canceled, the application is blocked until the requests have completed.

Not Applicable to PIOFS

When all file descriptors associated with a pipe or FIFO special file have been closed, any data remaining in the pipe or FIFO is discarded. If the link count of the file is 0 when all file descriptors associated with the file have been closed, the space occupied by the file is freed, and the file is no longer accessible.

Note: If the *FileDescriptor* parameter refers to a device and the **close** subroutine actually results in a device **close**, and the device **close** routine returns an error, the error is returned to the application. However, the *FileDescriptor* parameter is considered closed and it may not be used in any subsequent calls.

End of Not Applicable to PIOFS

All open file descriptors are closed when a process exits. In addition, file descriptors may be closed during the exec subroutine if the **close-on-exec** flag has been set for that file descriptor.

Parameters

FileDescriptor Specifies a valid open file descriptor.

Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the **errno** global variable is set to identify the error. If the **close** subroutine is interrupted by a signal that is caught, it returns a value of -1, the **errno** global variable is set to **EINTR** and the state of the *FileDescriptor* parameter is closed.

Error Codes

The **close** subroutine is unsuccessful if the following is true:

EBADF The *FileDescriptor* parameter does not specify a valid open file descriptor.

EINTR Specifies that the **close** subroutine was interrupted by a signal.

The **close** subroutine may also be unsuccessful if...

Not Applicable to PIOFS

...the file being closed is NFS-mounted and...

End of Not Applicable to PIOFS

...the server is down under the following conditions:

- The file is on a hard mount.
- The file is locked in any manner.

The **close** subroutine may also be unsuccessful if NFS is installed and the following is true:

ETIMEDOUT The connection timed out.

Related Information

Subroutines: **exec**, **fcntl**, **ioctl**, **lockfx**, **open**, **openx**, or **creat**, **pipe**, **socket**.

fclose or fflush

Purpose

fclose or fflush – Closes or flushes a stream.

Library

Standard C Library (**libc.a**)

C Syntax

#include <**stdio.h**>

int fclose(*Stream*)
FILE ***Stream*;**

int fflush(*Stream*)
FILE ***Stream*;**

Description

The **fclose** subroutine writes buffered data to the stream specified by the *Stream* parameter, and then closes the stream. The **fclose** subroutine is automatically called for all open files when the exit subroutine is invoked.

The **fflush** subroutine writes any buffered data for the stream specified by the *Stream* parameter and leaves the stream open.

If *Stream* is a null pointer, the **fflush** subroutine performs this flushing action on all streams for which the behavior is defined.

Parameters

Stream Specifies the output stream.

Return Values

Upon successful completion, the **fclose** and **fflush** subroutines return a value of 0. Otherwise, a value of EOF is returned.

Error Codes

If the **fclose** and **fflush** subroutines are unsuccessful, the following errors are returned through the **errno** global variable:

EAGAIN The **O_NONBLOCK** flag is set for the file descriptor under lying *Stream* and the process would be delayed in the write operation.

Not Applicable to PIOFS

EBADF The file descriptor underlying *Stream* is not valid.

EFBIG An attempt was made to write a file that exceeds the process' file size limit or the maximum file size. See the ulimit subroutine.

End of Not Applicable to PIOFS

EINTR	The **pfflush** subroutine was interrupted by a signal.
EIO	The process is a member of a background process group attempting to write to its controlling terminal, the **TOSTOP** signal is set, the process is neither ignoring nor blocking the **SIGTTOU** signal and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.
ENOSPC	There was no free space remaining on the device containing the file.

Not Applicable to PIOFS

EPIPE	An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A **SIGPIPE** signal is sent to the process.

End of Not Applicable to PIOFS

Related Information

The **close** subroutine, **exit**, **atexit**, or **_exit** subroutine, **fopen**, **freopen**, or **fdopen** subroutine, **setbuf**, **setvbuf**, **setbuffer**, or **setlinebuf** subroutine.

fcntl, dup2, or dup

Purpose

fcntl, **dup**, **dup2** – Controls open file descriptors

C Syntax

```
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

int fcntl (FileDescriptor,Command, Argument)
int FileDescriptor, Command, Argument;

int dup2 (Old, New)
int Old, New;

int dup (FileDescriptor)
int FileDescriptor;
```

Description

The **fcntl** subroutine performs controlling operations on the open file specified by the *FileDescriptor* parameter. If Network File System is installed on your system, the open file can reside on another node. The **fcntl** subroutine is used to:

- Duplicate open file descriptors.
- Set and get the file-descriptor flags.
- Set and get the file-status flags.

Not Applicable to PIOFS

- Manage record locks.

End of Not Applicable to PIOFS

- Manage asynchronous I/O ownership.
- Close multiple files.

The **fcntl** subroutine can provide the same functions as the **dup** and **dup2** subroutines. See "Compatibility Interfaces" on page 146.

If *FileDescriptor* refers to a terminal device or socket, then asynchronous I/O facilities can be used. These facilities are normally enabled by using the **ioctl** subroutine with the **FIOASYNC**, **FIOSETOWN**, and **FIOGETOWN** commands. However, a BSD-compatible mechanism is also available if the application is linked with the **libbsd.a** library.

When using the **libbsd.a** library, asynchronous I/O is enabled by using the **F_SETFL** command with the **FASYNC** flag set in the *Argument* parameter. The **F_GETOWN** and **F_SETOWN** commands get the current asynchronous I/O owner and set the asynchronous I/O owner.

Not Applicable to PIOFS

General Record Locking Information

A lock is either an *enforced* or *advisory* lock and either a *read* or a *write* lock.

Warning: Buffered I/O does not work properly when used with file locking. Do not use the standard I/O package routines on files that are going to be locked.

For a lock to be an enforced lock, the Enforced Locking attribute of the file must be set; for example, the **S_ENFMT** bit must be set, but the **S_IXGRP, S_IXUSR,** and **S_IXOTH** bits must be clear. Otherwise, the lock is an advisory lock. A given file can have advisory or enforced locks, but not both. The description of the **sys/mode.h** file includes a description of file attributes.

When a process holds an enforced lock on a section of a file, no other process can access that section of the file with the **read** or **write** subroutine. In addition, the **open** and **ftruncate** subroutines cannot truncate the locked section of the file, and the **fclear** subroutine cannot modify the locked section of the file. If another process attempts to read or modify the locked section of the file, the process either sleeps until the section is unlocked or returns with an error indication.

When a process holds an advisory lock on a section of a file, no other process can lock that section of the file (or an overlapping section) with the **fcntl** subroutine. (No other subroutines are affected.) As a result, processes must voluntarily call the **fcntl** subroutine in order to make advisory locks effective.

When a process holds a read lock on a section of a file, other processes can also set read locks on that section or on subsets of it. Read locks are also called *shared* locks.

A read lock prevents any other process from setting a write lock on any part of the protected area. If the read lock is also an enforced lock, no other process can modify the protected area.

The file descriptor on which a read lock is being placed must have been opened with read access.

When a process holds a write lock on a section of a file, no other process can set a read lock or a write lock on that section. Write locks are also called *exclusive* locks. Only one write lock and no read locks can exist for a specific section of a file at any time.

If the lock is also an enforced lock, no other process can read or modify the protected area.

The following general rules about file locking apply:

- Changing or unlocking part of a file in the middle of a locked section leaves two smaller sections locked at each end of the originally locked section.
- If the calling process holds a lock on a file, that lock can be replaced by later calls to the **fcntl** subroutine.

- All locks associated with a file for a given process are removed when the process closes *any* file descriptor for that file.
- Locks are not inherited by a child process after a fork subroutine is run.

Note: Deadlocks due to file locks in a distributed system are not always detected. When such deadlocks can possibly occur, the programs requesting the locks should set time-out timers.

Locks can start and extend beyond the current end of a file but cannot be negative relative to the beginning of the file. A lock can be set to extend to the end of the file by setting the **l_len** field to 0. If such a lock also has the **l_start** and **l_whence** fields set to 0, the whole file is locked. The **l_len, l_start,** and **l_whence** locking fields are part of the **flock** structure.

End of Not Applicable to PIOFS

Parameters

FileDescriptor Specifies an open file descriptor obtained from a successful call to the **open**, **fcntl**, or **pipe** subroutine. File descriptors are small positive integers used (instead of file names) to identify files.

Argument Specifies a variable whose value sets the function specified by the *Command* parameter.

Not Applicable to PIOFS

When dealing with file locks, the *Argument* parameter must be a pointer to the **FLOCK** structure.

End of Not Applicable to PIOFS

Command Specifies the operation performed by the **fcntl** subroutine. The **fcntl** subroutine can duplicate open file descriptors, set file-descriptor flags, set file descriptor locks, set process IDs, and close open file descriptors.

Duplicating File Descriptors

F_DUPFD Returns a new file descriptor as follows:

- Lowest-numbered available file descriptor greater than or equal to the *Argument* parameter
- Same object references as the original file
- Same file pointer as the original file (that is, both file descriptors share one file pointer if the object is a file)
- Same access mode (read, write, or read-write)
- Same file status flags (That is, both file descriptors share the same file status flags.)

- The **close-on-exec** flag (**FD_CLOEXEC** bit) associated with the new file descriptor is cleared

Setting File-Descriptor Flags

F_GETFD Gets the **close-on-exec** flag (**FD_CLOEXEC** bit) that is associated with the file descriptor specified by the *FileDescriptor* parameter. The *Argument* parameter is ignored. File descriptor flags are associated with a single file descriptor, and do not affect others associated with the same file.

F_SETFD Assigns the value of the *Argument* parameter to the **close-on-exec** flag (**FD_CLOEXEC** bit) that is associated with the *FileDescriptor* parameter. If the **FD_CLOEXEC** flag value is 0, the file remains open across any calls to **exec** subroutines; otherwise, the file will close upon the successful execution of an **exec** subroutine.

F_GETFL Gets the file-status flags and file-access modes for the open file description associated with the file descriptor specified by the *FileDescriptor* parameter. The open file description is set at the time the file is opened and applies only to those file descriptors associated with that particular call to the file. This open file descriptor does not affect other file descriptors that refer to the same file with different open file descriptions.

The file-status flags have the following values:

O_APPEND Set append mode.

Not Applicable to PIOFS

O_NONBLOCK No delay.

End of Not Applicable to PIOFS

The file-access modes have the following values:

O_RDONLY Open for reading only.

O_RDWR Open for reading and writing.

O_WONLY Open for writing only.

The file access flags can be extracted from the return value using the **O_ACCMODE** mask, which is defined in the **fcntl.h** file.

F_SETFL Sets the file status flags from the corresponding bits specified by the *Argument* parameter. The file-status flags are set for the open file description associated with the file descriptor specified by the *FileDescriptor* parameter. The following flags may be set:

- **O_APPEND** or **FAPPEND**

Not Applicable to PIOFS

- **O_NDELAY** or **FNDELAY**
- **O_NONBLOCK** or **FNONBLOCK**
- **O_SYNC** or **FSYNC**
- **FASYNC**

End of Not Applicable to PIOFS

The **O_NDELAY** and **O_NONBLOCK** flags affect only operations against file descriptors derived from the same **open** subroutine. In BSD, these operations apply to all file descriptors that refer to the object.

Setting File Descriptor Locks.

Not Applicable to PIOFS

F_GETLK Gets information on the first lock that blocks the lock described in the flock structure. The *Argument* parameter should be a pointer to a type **struct flock,** as defined in the **flock.h** file. The information retrieved by the **fcntl** subroutine overwrites the information in the **struct flock** pointed to by the *Argument* parameter. If no lock is found that would prevent this lock from being created, the structure is left unchanged, except for lock type (**l_type)** which is set to **F_UNLCK.**

F_SETLK Sets or clears a file-segment lock according to the lock description pointed to by the *Argument* parameter. The *Argument* parameter should be a pointer to a type **struct flock,** which is defined in the **flock.h** file. The **F_SETLK** option is used to establish read (or shared) locks (**F_RDLCK),** or write (or exclusive) locks (F_WRLCK), as well as to remove either type of lock (**F_UNLCK).** The lock types are defined by the **fcntl**.h file. If a shared or exclusive lock cannot be set, the **fcntl** subroutine returns immediately.

F_SETLKW Performs the same function as the **F_SETLK** option unless a read or write lock is blocked by existing locks, in which case the process sleeps until the section of the file is free to be locked. If a signal that is to be caught is received while the **fcntl** subroutine is waiting for a region, the **fcntl** subroutine is interrupted, returns a -1, sets the

errno global variable to **EINTR.** The lock operation is not done.

End of Not Applicable to PIOFS

Setting The Process ID

F_GETOWN Gets the process ID or process group currently receiving **SIGIO** and **SIGURG** signals. Process groups are returned as negative values.

F_SETOWN Sets the process or process group to receive **SIGIO** and **SIGURG** signals. Process groups are specified by supplying a negative *Argument* value. Otherwise, the *Argument* parameter is interpreted as a process ID.

Closing File Descriptors

F_CLOSEM Closes all file descriptors from FileDescriptor up to the number specified by the **OPEN_MAX** value.

Old Specifies an open file descriptor.

New Specifies an open file descriptor that is returned by the **dup2** subroutine.

Compatibility Interfaces

Not Applicable to PIOFS

The lockfx Subroutine

The **fcntl** subroutine functions similar to the **lockfx** subroutine, when the *Command* parameter is **F_SETLK, F_SETLKW,** or **F_GETLK,** and when used in the following way:

```
fcntl (FileDescriptor, Command, Argument)
```

is equivalent to:

```
lockfx (FileDescriptor, Command, Argument)
```

End of Not Applicable to PIOFS

The dup and dup2 Subroutines

The **fcntl** subroutine functions similar to the **dup** and **dup2** subroutines, when used in the following way:

```
dup (FileDescriptor)
```

is equivalent to:

```
fcntl (FileDescriptor, F_DUPFD, 0)
```

```
dup2 (Old, New)
```

is equivalent to:

```
close (New);
fcntl(Old, F_DUPFD, New)
```

The **dup** and **dup2** subroutines differ from the **fcntl** subroutine in the following ways:

- If the file descriptor specified by the *New* parameter is greater than or equal to **OPEN_MAX**, the **dup2** subroutine returns a -1 and sets the **errno** variable to **EBADF.**
- If the file descriptor specified by the *Old* parameter is valid and equal to the file descriptor specified by the *New* parameter, the **dup2** subroutine will return the file descriptor specified by the *New* parameter, without closing it.
- If the file descriptor specified by the *Old* parameter is not valid, the **dup2** subroutine will be unsuccessful and will not close the file descriptor specified by the *New* parameter.
- The value returned by the **dup** and **dup2** subroutines is equal to the *New* parameter upon successful completion; otherwise, the return value is -1.

Return Values

Upon successful completion, the value returned depends on the value of the *Command* parameter, as follows:

F_DUPFD A new file descriptor

F_GETFD The value of the flag (only the **FD_CLOEXEC** bit is defined)

F_SETFD A value other than -1

F_GETFL The value of file flags

F_SETFL A value other than -1

F_GETOWN The value of descriptor owner

F_SETOWN A value other than -1

Not Applicable to PIOFS

F_GETLK A value other than -1

F_SETLK A value other than -1

F_SETLKW A value other than -1

End of Not Applicable to PIOFS

F_CLOSEM A value other than -1.

If the **fcntl** subroutine fails, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **fcntl** subroutine is unsuccessful if one or more of the following are true:

EBADF The *FileDescriptor* parameter is not a valid open file descriptor.

EMFILE The *Command* parameter is **F_DUPFD,** and the maximum number of file descriptors are currently open (**OPEN_MAX).**

EINVAL The *Command* parameter is **F_DUPFD,** and the Argument parameter is negative or greater than or equal to **OPEN_MAX.**

EINVAL An illegal value was provided for the *Command* parameter.

EINVAL An attempt was made to lock a fifo or pipe.

ESRCH The value of the *Command* parameter is **F_SETOWN,** and the process ID specified as the *Argument* parameter is not in use.

EINTR The *Command* parameter was **F_SETLKW** and the process received a signal while waiting to acquire the lock.

The **dup** and **dup2** subroutines fail if one or both of the following are true:

EBADF The *Old* parameter specifies an invalid open file descriptor or the *New* parameter specifies a file descriptor that is out of range.

EMFILE The number of file descriptors exceeds the **OPEN_MAX** value or there is no file descriptor above the value of the *New* parameter.

Not Applicable to PIOFS

If NFS is installed on the system, the **fcntl** subroutine can fail if the following is true:

End of Not Applicable to PIOFS

ETIMEDOUT The connection timed out.

Related Information

Commands: **close, fork, ioctl. ioctlx, execl, excecv, execle, execve, execlp, execvp, exect, lockf, openx, open, creat, tines, read, write.**

fopen, freopen, or fdopen

Purpose

fopen, **freopen**, or **fdopen** – Opens a stream.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <stdio.h>

FILE *fopen (
        const char *Path,
        const char *Type );

FILE *freopen (
        char *Path,
        char *Type,
        FILE *Stream );

FILE *fdopen (
        int FileDescriptor,
        const char *Type);
```

Parameters

Path Points to a character string that contains the name of the file to be opened.

Type Points to a character string that has one of the following values:

r Opens a text file for reading.

w Creates a new text file for writing, or opens and truncates a file to 0 length.

Not Applicable to PIOFS

a Appends (opens a text file for writing at the end of the file, or creates a file for writing).

End of Not Applicable to PIOFS

rb Opens a binary file for reading.

wb+ Creates a binary file for writing, or opens and truncates a file to 0.

Not Applicable to PIOFS

ab Appends (opens a binary file for writing at the end of the file, or creates a file for writing).

End of Not Applicable to PIOFS

r+ Opens a file for update (reading and writing).

w+ Truncates or creates a file for update.

Not Applicable to PIOFS

a+ Appends (opens a text file for writing at end of file, or creates a file for writing).

End of Not Applicable to PIOFS

r+b, **rb+** Opens a binary file for update (reading and writing).

w+b, **wb+** Creates a binary file for update, or opens and truncates a file to 0 length.

Not Applicable to PIOFS

a+b, **ab+** Appends (opens a binary file for update, writing at the end of the file, or creates a file for writing).

End of Not Applicable to PIOFS

Note: The operating system does not distinguish between text and binary files. The **b** value in the *Type* parameter value is ignored.

Stream Specifies the input stream.

FileDescriptor Specifies a valid open file descriptor.

Description

The **fopen** subroutine opens the file named by the *Path* parameter and associates a stream with it. The **fopen** subroutine returns a pointer to the **FILE** structure of this stream.

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without an intervening **fflush** subroutine call or a file positioning operation (**fseek**, **fsetpos**, or **rewind** subroutine). Also, an input operation cannot be directly followed by an output operation without an intervening flush or file positioning operation, unless the input operation encounters the end of the file.

Not Applicable to PIOFS

When you open a file for appending (that is, when the *Type* parameter is set to **a** or **a+**), it is impossible to overwrite information already in the file. You can, however, use **a+** to update a file anywhere after the original contents. When the *Type* parameter is set to **w+** it is possible to overwrite existing information anywhere in the file. You can use the **fseek** subroutine to change the location of the file pointer to any position in the file, but when output is written to the file, the current file

pointer is ignored. All output is written at the end of the file and the file pointer's position changes to the end of the output.

If two separate processes open the same file for append, each process can write freely to the file without destroying the output being written by the other. The output from the two processes is intermixed in the order in which it is written to the file.

End of Not Applicable to PIOFS

Note: If the data is buffered, it is not actually written until it is flushed.

The **freopen** subroutine substitutes the named file in place of the open stream. The original stream is closed regardless of whether the **openx** subroutine succeeds. The **freopen** subroutine returns a pointer to the **FILE** structure associated with *Stream.* The **freopen** subroutine is typically used to attach the pre-opened streams associated with standard input (**stdin**), standard output (**stdout**), and the **stderr** file to other files.

The **fdopen** subroutine associates a stream with a file descriptor obtained from an **openx** subroutine, **dup** subroutine, **creat** subroutine, or **pipe** subroutine. These subroutines open files but do not return pointers to **FILE** structures. Many of the standard I/O package subroutines require pointers to **FILE** structures.

The *Type* parameter for the **fdopen** subroutine specifies the mode of the stream, such as **r** to open a file for reading,

Not Applicable to PIOFS

or **a** to open a file for appending (writing at the end of the file).

End of Not Applicable to PIOFS

The mode value of the *Type* parameter specified with the **fdopen** subroutine must agree with the mode of the file specified when the file was originally opened or created.

Return Values

If the **fdopen**, **fopen** or **freopen** subroutine is unsuccessful, a null pointer is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **fopen** and **freopen** subroutines are unsuccessful if the following is true:

EACCES Search permission is denied on a component of the path prefix, the file exists and the permissions specified by the mode are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

EINTR A signal was received during the process.

EISDIR The named file is a directory and the process does not have write access to it.

Not Applicable to PIOFS

ENAMETOOLONG The file name or path name is too long and it cannot be truncated.

End of Not Applicable to PIOFS

ENFILE The maximum number of files allowed are currently open.

ENOENT The named file does not exist or the *FileDescriptor* parameter points to an empty string.

ENOSPC The file is not yet created and the directory or file system to contain the new file cannot be expanded.

ENOTDIR A component of the path prefix is not a directory.

ENXIO The named file is a character-special or block-special file, and the device associated with this special file does not exist.

Not Applicable to PIOFS

EROFS The named file resides on a read-only file system and does not have write access.

End of Not Applicable to PIOFS

ETXTBSY The file is a pure-procedure (shared-text) file that is being executed and the process does not have write access.

The **fdopen**, **fopen**, or **freopen** subroutines are unsuccessful if the following is true:

EINVAL The value of the *Type* parameter is not valid.

EINVAL A keyword has been specified incorrectly in the **piofs.profile** file.

EMFILE FOPEN_MAX streams are currently open in the calling process.

ENOMEM Insufficient storage space is available.

The **fdopen** subroutine is unsuccessful if the following is true:

EBADF The value of the *FileDescriptor* parameter is not valid.

POSIX

w Truncates to 0 length or creates text file for writing.

w+ Truncates to 0 length or creates text file for update.

Not Applicable to PIOFS

a Opens or creates text file for writing at end of file.

a+ Opens or creates text file for update, writing at end of file.

End of Not Applicable to PIOFS

Related Information

The **fclose** or **fflush** subroutine, **fseek**, **rewind**, **ftell**, **fgetpos**, or **fsetpos** subroutine, **open**, **openx**, or **creat** subroutine, **setbuf**, **setvbuf**, **setbuffer**, or **setlinebuf** subroutine.

fread or fwrite

Purpose

Reads and writes binary files.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <stdio.h>

size_t fread (
        (void *) Pointer,
        size_t Size,
        size_t NumberOfItems,
        FILE *Stream );

size_t fwrite (
        (void *) Pointer,
        size_t Size,
        size_t NumberOfItems,
        FILE *Stream );
```

Parameters

Pointer	Points to an array.
Size	Specifies the size of the variable type of the array pointed to by the *Pointer* parameter. The *Size* parameter can be considered the same as a call to **sizeof** subroutine.
NumberOfItems	Specifies the number of items of data.
Stream	Specifies the input or output stream.

Description

The **fread** subroutine copies the number of data items specified by the *NumberOfItems* parameter from the input stream into an array beginning at the location pointed to by the *Pointer* parameter. Each data item has the type **Pointer*.

The **fread** subroutine stops copying bytes if an end-of-file (EOF) or error condition is encountered while reading from the input specified by the *Stream* parameter, or when the number of data items specified by the *NumberOfItems* parameter have been copied. This subroutine leaves the file pointer of the *Stream* parameter, if defined, pointing to the byte following the last byte read. The **fread** subroutine does not change the contents of the *Stream* parameter.

Note: The **fread** subroutine is a buffered read subroutine library call. It reads data in 4KB blocks. For tape block sizes greater than 4KB, use the open subroutine and read subroutine.

The **fwrite** subroutine appends to the output stream the contents of the *NumberOfItems* parameter of data of the type **Pointer* from the array pointed to by the *Pointer* parameter.

The **fwrite** subroutine appends items to the output stream from the array pointed to by *Pointer*. The **fwrite** subroutine appends as many items as specified in the *NumberOfItems* parameter.

The **fwrite** subroutine stops writing bytes if an error condition is encountered on the stream, or when the number of items of data specified by the *NumberOfItems* parameter have been written. The **fwrite** subroutine does not change the contents of the array pointed to by the *Pointer* parameter.

Return Values

The **fread** and **fwrite** subroutines return the number of items actually transferred. If the *NumberOfItems* parameter contains a 0, no characters are transferred, and a value of 0 is returned. If the *NumberOfItems* parameter contains a negative number, it is translated to a positive number, since the *NumberOfItems* parameter is of type unsigned.

Error Codes

The **fread** subroutine is unsuccessful if the I/O stream is unbuffered or data needs to be read into the I/O stream's buffer, and one or more of the following are true:

EAGAIN	The O_NONBLOCK flag is set for the file descriptor specified by the *Stream* parameter, and the process would be delayed in the **fread** operation.
EBADF	The file descriptor specified by the *Stream* parameter is not a valid file descriptor open for reading.
EINTR	The read operation was terminated due to receipt of a signal and no data was transferred.
EIO	The process is a member of a background process group attempting to perform a read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group has no parent process.
ENOMEM	Insufficient storage space is available.
ENXIO	A request was made of a nonexistent device.

The **fwrite** subroutine is unsuccessful if either the I/O stream is unbuffered or the I/O stream's buffer needs to be flushed, and one or more of the following are true:

EAGAIN	The O_NONBLOCK flag is set for the file descriptor specified by the *Stream* parameter, and the process is delayed in the write operation.
EBADF	The file descriptor specified by the *Stream* parameter is not a valid file descriptor open for writing.

Not Applicable to PIOFS

EFBIG	An attempt was made to write a file that exceeds the file size of the process limit or the systemwide maximum file size.

End of Not Applicable to PIOFS

EINTR	The write operation was terminated due to the receipt of a signal, and no data was transferred.
EIO	The process is a member of a background process group attempting to perform a write to its controlling terminal, the TOSTOP signal is set, the process is neither ignoring nor blocking the SIGTTOU signal, and the process group of the process is orphaned.
ENOSPC	There was no free space remaining on the device containing the file.

Not Applicable to PIOFS

EPIPE	An attempt is made to write to a pipe or FIFO process that is not open for reading by any process. A SIGPIPE signal is sent to the process.

End of Not Applicable to PIOFS

The **fwrite** subroutine is unsuccessful if either of the following are true:

ENOMEM	Insufficient storage space is available.
ENXIO	A request was made of a nonexistent device, or the request was outside the capabilities of the device.

Related Information

The **fopen**, **freopen**, or **fdopen** subroutine, **getc**, **fgetc**, **getchar**, or **getw** subroutine, **getwc**, **fgetwc**, or **getwchar** subroutine, **gets** or **fgets** subroutine, **getws** or **fgetws** subroutine, **print**, **fprintf**, or **sprintf** subroutine, **putc**, **putchar**, **fputc**, or **putw** subroutine, **putwc**, **putwchar**, or **fputwc** subroutine, **puts** or **fputs** subroutine, **putws** or **fputws** subroutine, **read** subroutine, **scanf**, **fscanf**, **sscanf**, or **wsscanf** subroutine, **write** subroutine.

fseek, rewind, ftell, fgetpos, or fsetpos

Purpose

fseek, **ftell**, **fgetpos**, or **fsetpos** – Repositions the file pointer of a stream.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <stdio.h>

int fseek (
        FILE *Stream,
        long int Offset,
        int Whence );

void rewind ( FILE *Stream );

long int ftell ( FILE *Stream );

int fsetpos (
        FILE *Stream,
        const fpos_t *Position );

int fgetpos (
        FILE *Stream,
        fpos_t *Position );
```

Parameters

Stream	Specifies the I/O stream.
Offset	Determines the position of the next operation.
Whence	Determines the value for the file pointer associated with the *Stream* parameter.
Position	Specifies the value of the file-position indicator.

Description

The **fseek** subroutine sets the position of the next input or output operation on the I/O stream specified by the *Stream* parameter. The position of the next operation is determined by the *Offset* parameter, which can be either positive or negative.

The **fseek** subroutine sets the file pointer associated with the specified *Stream* as follows:

- If the *Whence* parameter is set to 0, the pointer is set to the value of the *Offset* parameter.
- If the *Whence* parameter is set to 1, the pointer is set to its current location plus the value of the *Offset* parameter.
- If the *Whence* parameter is set to 2, the pointer is set to the size of the file plus the value of the *Offset* parameter.

The **fseek** subroutine is unsuccessful if attempted on a file that has not been opened using the **fopen** subroutine. In particular, the **fseek** subroutine cannot be used on a terminal or on a file opened with the **popen** subroutine.

The **rewind** subroutine is equivalent to calling the **fseek** subroutine using parameter values of (*Stream*,0,0), except that the **rewind** subroutine does not return a value.

The **fseek** and **rewind** subroutines undo any effects of the **ungetc** and **ungetwc** subroutines.

A successful call to the **fsetpos** subroutine clears the EOF indicator and undoes any effects of the **ungetc** and **ungetwc** subroutines.

After an **fseek** or a **rewind** subroutine, the next operation on a file opened for update can be either input or output.

The **fgetpos** subroutine is similar to the **ftell** subroutine, and the **fsetpos** subroutine is similar to the **fseek** subroutine. The **fgetpos** subroutine stores the current value of the file-position indicator for the stream pointed to by the *Stream* parameter in the object pointed to by the *Position* parameter. The **fsetpos** subroutine sets the file position indicator according to the value of the *Position* parameter, returned by a prior call to the **fgetpos** subroutine.

Applicable only to PIOFS

Note: If you use the Parallel I/O File System and specify USE_PIOFS_LSEEK in your **piofs.profile** file to disable the use of the system file pointer, these subroutines (and any others that reference the system file pointer) have unpredictable results. See “Accessing PIOFS Files Correctly” on page 92 for more information.

End of Applicable only to PIOFS

Return Values

Upon successful completion, the **fseek** subroutine returns a value of 0. Otherwise, a nonzero value is returned.

Upon successful completion, the **ftell** subroutine returns the offset of the current byte relative to the beginning of the file associated with the named stream. Otherwise, a long int value of -1 is returned and the **errno** global variable is set.

Upon successful completion, the **fgetpos** and **fsetpos** subroutines return a value of 0. Otherwise, a nonzero value is returned and the **errno** global variable is set to the specific error.

The **errno** global variable is used to determine if an error occurred during a **rewind** subroutine call.

Error Codes

The **fseek**, **ftell**, and **rewind** subroutines are unsuccessful if either the stream is unbuffered or the stream buffer needs to be flushed, and the call to the subroutine causes an underlying **lseek** or **write** subroutine to be invoked:

EAGAIN — The O_NONBLOCK flag is set for the file descriptor, delaying the process in the write operation.

EBADF — The file descriptor underlying the *Stream* parameter is not open for writing.

Not Applicable to PIOFS

EFBIG — An attempt has been made to write to a file that exceeds the file-size limit of the process or the maximum file size.

End of Not Applicable to PIOFS

EINTR — The write operation has been terminated because the process has received a signal, and either no data was transferred, or the implementation does not report partial transfers for this file.

EIO — The process is a member of a background process group attempting to perform a **write** subroutine to its controlling terminal, the TOSTOP flag is set, the process is not ignoring or blocking the SIGTTOU signal, and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.

ENOSPC — There is no remaining free space on the device containing the file.

Not Applicable to PIOFS

EPIPE — An attempt has been made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

End of Not Applicable to PIOFS

EINVAL — The *Whence* parameter is not valid. The resulting file-position indicator will be set to a negative value. The EINVAL error code does not apply to the **ftell** and **rewind** subroutines.

Not Applicable to PIOFS

ESPIPE — The file descriptor underlying the *Stream* parameter is associated with a pipe or FIFO.

End of Not Applicable to PIOFS

The **fgetpos** and **fsetpos** subroutines are unsuccessful under the following conditions:

EINVAL	Either the *Stream* or the *Position* parameter is not valid.
EBADF	The file descriptor underlying the *Stream* parameter is not open for writing.

Not Applicable to PIOFS

ESPIPE	The file descriptor underlying the *Stream* parameter is associated with a pipe or FIFO.

End of Not Applicable to PIOFS

Related Information

The **fopen**, **freopen**, or **fdopen** subroutine, **lseek** subroutine, **opendir**, **readdir**, **telldir**, **seekdir**, **rewinddir**, or **closedir** subroutine, **popen** subroutine, **ungetc** or **ungetwc** subroutine, **write**, **writex**, **writev**, or **writevx** subroutine.

getc, getchar, fgetc, or getw

Purpose

Gets a character or word from an input stream.

Library

Standard I/O Package (**libc.a**)

C Syntax

#include <**stdio.h**>

int getc (FILE ****Stream* **);

int fgetc (FILE ****Stream* **);

int getchar (void);

int getw (FILE ****Stream* **);

Parameters

Stream	A pointer to the file structure of an open file.

Description

The **getc** macro returns the next byte from the input specified by the *Stream* parameter and moves the file pointer, if defined, ahead one byte in *Stream*. The **getc** macro cannot be used where a subroutine is necessary; for example, a subroutine pointer cannot point to it.

Because it is implemented as a macro, the **getc** macro does not work correctly with a *Stream* parameter that has side effects. In particular, the following does not work:

```
getc(*f++);
```

In cases like this, use the **fgetc** subroutine instead.

The **fgetc** subroutine performs the same function as the **getc** macro, but **fgetc** is a genuine subroutine, not a macro. The **fgetc** subroutine runs more slowly than **getc** but takes less space.

The **getchar** macro returns the next byte from **stdin** (the standard input stream). The **getchar** macro is equivalent to **getc(stdin)**.

The **getc** and **getchar** macros have also been implemented as subroutines for ANSI compatibility. To access the subroutines instead of the macros, insert **#undef getc** or **#undef getchar** at the beginning of the source file.

The **getw** subroutine returns the next word (**int**) from the input specified by the *Stream* parameter and increments the associated file pointer, if defined, to point to the next word. The size of a word varies from one machine architecture to another. The **getw** subroutine returns the constant EOF at the end of the file or when an error occurs. Since EOF is a valid integer value, the **feof** and **ferror** subroutines

should be used to check the success of **getw**. The **getw** subroutine assumes no special alignment in the file.

Because of additional differences in word length and byte ordering from one machine architecture to another, files written using the **putw** subroutine are machine-dependent and may not be readable using the **getw** macro on a different type of processor.

Return Values

Upon successful completion, the **getc**, **fgetc**, **getchar**, and **getw** subroutines return the next byte or integer from the input stream pointed by the *Stream* parameter. If the stream is at the end-of-file, an end-of-file indicator is set for the stream and the integer constant EOF is returned. If a read error occurs, the **errno** global variable is set to reflect the error, and EOF is returned. The **ferror** and **feof** subroutines should be used to distinguish between the end of the file and an error condition.

Error Codes

The subroutines will be unsuccessful if either the stream specified by the *Stream* parameter is unbuffered or data needs to be read into the stream's buffer, and:

EAGAIN — The O_NONBLOCK flag is set for the file descriptor underlying the stream specified by the *Stream* parameter. The process would be delayed in the **fgetc** subroutine operation.

EBADF — The file descriptor underlying the stream specified by the *Stream* parameter is not a valid file descriptor opened for reading.

Not Applicable to PIOFS

EFBIG — An attempt was made to read a file that exceeds the process's file-size limit or the maximum file size. See the **ulimit** subroutine.

End of Not Applicable to PIOFS

EINTR — The read operation was terminated due to the receipt of a signal, and either no data was transferred, or the implementation does not report partial transfer for this file.

Note: Depending upon which library routine the application binds to, this subroutine may return EINTR. Refer to the **signal** subroutine regarding the **SA_RESTART** bit.

EIO — The process is in a background process group attempting to perform a **read** subroutine call from its controlling terminal, and either the process is ignoring (or blocking) the SIGTTIN signal or the process group is orphaned.

Not Applicable to PIOFS

EPIPE — An attempt is made to read from a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

End of Not Applicable to PIOFS

The subroutines will also be unsuccessful under the following conditions:

ENOMEM	Indicates insufficient storage space is available.
ENXIO	Indicates either a request was made of a nonexistent device or the request was outside the capabilities of the device.

Related Information

The **feof**, **ferror**, **clearerr**, or **fileno** subroutine, **fopen**, **freopen**, or **fdopen** subroutine, **fread** or **fwrite** subroutine, **getwc**, **fgetwc**, or **getwchar** subroutine, **gets** or **fgets** subroutine, **putc**, **putchar**, **fputc**, or **putw** subroutine, **scanf**, **fscanf**, **sscanf**, or **wsscanf** subroutine.

llseek (AIX Version 4.1)

Purpose

llseek – Moves the read-write file pointer.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <unistd.h>
#include <sys/limits.h>

offset_t llseek (FileDescriptor, Offset, Whence)
int FileDescriptor, Whence;
offset_t Offset;
```

Description

The **llseek** subroutine sets the read-write file pointer for the open file specified by the **FileDescriptor** parameter.

Not Applicable to PIOFS

The **llseek** subroutine limits the read-write file pointer *Offset* to **OFF_MAX** for non-special files. For block and character special files, the **llseek** subroutine limits file pointer *Offset* to **DEV_OFF_MAX**.

End of Not Applicable to PIOFS

When an application restricts its name space to **ANSI_C_SOURCE**, **POSIX_SOURCE**, or **XOPEN_SOURCE**, or compiles with langlvl-ansi then the symbols **offset_t** and **DEV_OFF_MAX** are not defined and the **llseek** interface is not available.

Parameters

FileDescriptor Specifies a file descriptor obtained from a successful **open** or **fcntl** subroutine.

Offset Specifies a value, in bytes, that is used in conjunction with the *Whence* parameter to set the file pointer. A negative value causes seeking in the reverse direction.

Whence Specifies how to interpret the *Offset* parameter by setting the file pointer associated with the *FileDescriptor* parameter to one of the following variables:

SEEK_SET Sets the file pointer to the value of the *Offset* parameter.

SEEK_CUR Sets the file pointer to its current location plus the value of the *Offset* parameter.

SEEK_END Sets the file pointer to the size of the file plus the value of the *Offset* parameter.

Applicable only to PIOFS

This parameter is valid only if the file is opened as a single subfile.

End of Applicable only to PIOFS

Return Values

Upon successful completion, the resulting pointer location, measured in bytes from the beginning of the file, is returned. If the **llseek** subroutine is unsuccessful, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **llseek** subroutine is unsuccessful and the file pointer remains unchanged if any of the following are true:

EBADF The *FileDescriptor* parameter is not an open file descriptor.

Not Applicable to PIOFS

ESPIPE The *FileDescriptor* parameter is associated with a pipe (FIFO) or a socket.

End of Not Applicable to PIOFS

EINVAL The *Whence* parameter is an invalid value.

EINVAL The resulting offset would be negative.

Not Applicable to PIOFS

EINVAL The resulting offset would exceed **OFF_MAX** on any regular file.

EINVAL The resulting offset would exceed **DEV_OFF_MAX** on a block or character special file.

End of Not Applicable to PIOFS

Related Information

Subroutines: **fcntl, fseek, rewind, ftell, fgetpos, fsetpos, open, openx, creat, read, readx, readv, readvx, write, writex, writev, writevx**.

lseek

Purpose

lseek – Moves the read-write file pointer.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <unistd.h>

off_t lseek (FileDescriptor, Offset, Whence)
int FileDescriptor, Whence;
off_t Offset;
```

Description

Not Applicable to PIOFS

The **lseek** subroutine sets the read-write file pointer for the open file specified by the **FileDescriptor** parameter.

End of Not Applicable to PIOFS

Applicable only to PIOFS

The **lseek** subroutine sets the read-write file pointer for the open subfile specified by the **FileDescriptor** parameter.

If the PIOFS file was opened using a **piofs.profile** with USE_PIOFS_LSEEK specified, do not use **lseek** because the Parallel I/O File System uses its own internal 64-bit offset and ignores the offset set by **lseek**. See "Accessing PIOFS Files Correctly" on page 92 for more more information.

See Table 4 on page 93 for information on how to change the PIOFS file offset.

End of Applicable only to PIOFS

Parameters

FileDescriptor	Specifies a file descriptor obtained from a successful **open** or **fcntl** subroutine.
Offset	Specifies a value, in bytes, that is used in conjunction with the *Whence* parameter to set the file pointer. A negative value causes seeking in the reverse direction.
Whence	Specifies how to interpret the *Offset* parameter by setting the file pointer associated with the *FileDescriptor* parameter to one of the following variables:

SEEK_SET Sets the file pointer to the value of the *Offset* parameter.

SEEK_CUR Sets the file pointer to its current location plus the value of the *Offset* parameter.

SEEK_END Sets the file pointer to the size of the file plus the value of the *Offset* parameter.

Applicable only to PIOFS

This parameter is valid only if the file is opened as a single subfile.

End of Applicable only to PIOFS

Return Values

Upon successful completion, the resulting pointer location, measured in bytes from the beginning of the file, is returned. If the **lseek** subroutine is unsuccessful, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **lseek** subroutine is unsuccessful and the file pointer remains unchanged if any of the following are true:

EBADF The *FileDescriptor* parameter is not an open file descriptor.

Not Applicable to PIOFS

ESPIPE The *FileDescriptor* parameter is associated with a pipe (FIFO) or a socket.

End of Not Applicable to PIOFS

EINVAL The *Whence* parameter is an invalid value.

EINVAL The resulting offset would be negative.

Related Information

Subroutines:

fcntl, fseek, rewind, ftell, fgetpos, fsetpos, open, openx, creat, read, readx, readv, readvx, write, writex, writev, writevx.

mkdir

Purpose

mkdir – Creates a directory.

Library

Standard C Library (**libc.a**)

C Syntax

#include <**sys**/**stat.h**>

int mkdir (*Path***,** *Mode***)**
char **Path***Path***;**
int *Mode***;**

Description

The **mkdir** subroutine creates a new directory.

The new directory has the following:

- The owner ID is set to the process-effective user ID.
- If the parent directory has the **SetFileGroupID** (**S_ISGID)** attribute set, the new directory inherits the group ID of the parent directory. Otherwise, the group ID of the new directory is set to the effective group ID of the calling process.
- Permission and attribute bits are set according to the value of the *Mode* parameter, with the following modifications:
  - All bits set in the process-file mode-creation mask are cleared.
  - The **SetFileUserID** and **Sticky** (**S_ISVTX)** attributes are cleared.
- If the **Path** variable names a symbolic link, the link is followed. The new directory is created where the variable pointed.

Parameters

Path Specifies the name of the new directory. If Network File System (NFS) is installed on your system, this path can cross into another node. In this case, the new directory is created at that node.

To execute the **mkdir** subroutine, a process must have search permission to get to the parent directory of the *Path* parameter as well as write permission in the parent directory itself.

Mode Specifies the mask for the read, write, and execute flags for owner, group, and others. The *Mode* parameter specifies directory permissions and attributes. This parameter is costructed by logically ORing values described in the **sys**/**mode.h** file.

Return Values

Upon successful completion, the **mkdir** subroutine returns a value of 0. Otherwise, a value of -1 is returned, and the **errno** global variable is set to indicate the error.

Error Codes

The **mkdir** subroutine is unsuccessful and the directory is not created if one or more of the following are true:

EACCES Creating the requested directory requires writing in a directory with a mode that denies write permission.

EEXIST The named file already exists.

Not Applicable to PIOFS

EROFS The named file resides on a read-only file system.

End of Not Applicable to PIOFS

ENOSPC The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

Not Applicable to PIOFS

EMLINK The link count of the parent directory exceeds the maximum.

End of Not Applicable to PIOFS

Not Applicable to PIOFS

ENAMETOOLONG The **Path** parameter or a path component is too long and cannot be truncated.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG The **Path** parameter exceeded 255 characters or a path component exceeded 255 characters.

End of Applicable only to PIOFS

ENOENT A component of the path prefix does not exist or the **Path** parameter points to an empty string.

ENOTDIR A component of the path prefix is not a directory.

Not Applicable to PIOFS

EDQUOT The directory in which the entry for the new directory is being placed cannot be extended, or an i-node or disk blocks could not be allocated for the new directory because the user's or group's quota of disk blocks or i-nodes on the file system containing the directory is exhausted.

End of Not Applicable to PIOFS

The **mkdir** subroutine can be unsuccessful for other reasons. See "Error Codes for Services that Require Path Name Resolution" on page 125 for a list of additional errors.

If NFS is installed on the system, the **mkdir** subroutine is also unsuccessful if the following is true:

ETIMEDOUT The connection timed out.

Related Information

Commands: **command, mkdir mknod**.

Subroutines: **chmod**, **mknod**, **rmdir**, **umask**.

open, openx, creat

Purpose

open, openx, creat – Opens a file for reading or writing

C Syntax

#include <**fcntl.h**>

int open (*Path***, *OFlag*, [***Mode***])**
const char ****Path***;
int *OFlag*;
mode_t *Mode*;

int openx (*Path***, *OFlag*, *Mode*, *Extension***)**
const char ****Path***;
int *OFlag*;
mode_t *Mode*;
int *Extension*;

int creat(*Path***, [***Mode***])**
const char ****Path***;
mode_t *Mode*;

Description

The **open**, **openx**, and **creat** subroutines establish a connection between the file named by the *Path* parameter and a file descriptor. The opened file descriptor is used by subsequent I/O subroutines, such as **read** and **write**, to access that file.

The **openx** subroutine is the same as the **open** subroutine, with the addition of an *Extension* parameter, which is provided for device driver use. The **creat** subroutine is equivalent to the **open** subroutine with the **O_WRONLY**, **O_CREAT** and **O_TRUNC** flags set.

The returned file descriptor is the lowest file descriptor not previously open for that process. No process can have more than **OPEN_MAX** file descriptors open simultaneously.

The file offset, marking the current position within the file, is set to the beginning of the file. The new file descriptor is set to remain open across **exec** subroutines.

Parameters

Path	Specifies the file to be opened.
Mode	Specifies the read, write, and execute permissions of the file to be created (requested by the **O_CREAT flag**). If the file already exists, this parameter is ignored.
Extension	Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the *Extension* parameter in a device-dependent way, either as a value or as a pointer to a communication area.

Drivers must apply reasonable defaults when the *Extension* parameter value is 0.

OFlag Specifies the type of access, special open processing, the type of update, and the initial state of the open file. The parameter value is constructed by logically ORing special open processing flags. These flags are defined in the **fcntl.h** file and are described in the following flags.

Oflag Parameter Flag Values That Specify Type of Access

The following OFlag parameter flag values specify type of access:

O_RDONLY The file is opened for reading only.

O_WRONLY The file is opened for writing only.

O_RDWR The file is opened for both reading and writing.

Note: One of the file access values must be specified. Do not use **O_RDONLY**, **O_WRONLY**, or **O_RDWR** together. If none is set, none is used, and the results are unpredictable.

Oflag Parameter Values That Specify Special Open Processing

The following **OFlag** parameter flag values specify special open processing:

O_CREAT If the file exists, this flag has no effect, except as noted under the **O_EXCL** flag. If the file does not exist, a regular file is created with the following characteristics:

- The owner ID of the file is set to the effective user ID of the process.
- The group ID of the file is set to the group ID of the parent directory if the parent directory has the **SetGroupID** attribute (**S_ISGID bit**) set. Otherwise, the group ID of the file is set to the effective group ID of the calling process.
- The file permission and attribute bits are set to the value of the *Mode* parameter, modified as follows:
  - All bits set in the process file mode creation mask are cleared. (The file creation mask is described in the **umask** subroutine.)
  - The S_ISVTX attribute bit is cleared.

O_EXCL If the **O_EXCL** and **O_CREAT** flags are set, the open is unsuccessful if the file exists.

Not Applicable to PIOFS

Note: The **O_EXCL** flag is not fully supported for Network File Systems (NFS). The NFS protocol does not guarantee the designed function of the **O_EXCL** flag.

O_NSHARE Assures that no process has this file open and precludes subsequent opens. If the file is on a physical file system and is already open, this open is unsuccessful and returns immediately

unless the *OFlag* parameter also specifies the **O_DELAY** flag. This flag is effective only with physical file systems.

Note: This flag is not supported by NFS.

O_RSHARE Assures that no process has this file open for writing and precludes subsequent opens for writing. The calling process can request write access. If the file is on a physical file system and is open for writing or open with the **O_NSHARE** flag, this open fails and returns immediately unless the *OFlag* parameter also specifies the **O_DELAY** flag.

Note: This flag is not supported by NFS.

O_DEFER The file is opened for deferred update. Changes to the file are not reflected on permanent storage until an **fsync** subroutine operation is performed. If no **fsync** subroutine operation is performed, the changes are discarded when the file is closed.

Note: This flag is not supported by NFS.

O_NOCTTY This flag specifies that the controlling terminal should not be assigned during this open.

End of Not Applicable to PIOFS

O_TRUNC If the file does not exist, this flag has no effect. If the file exists, is a regular file, and is successfully opened with the **O_RDWR** flag or the **O_WRONLY** flag, all of the following apply:

- The length of the file is truncated to 0.
- The owner and group of the fileare unchanged.
- The **SetUserID** attribute of the file mode is cleared.
- The **SetUserID** attribute of the file is cleared.

The **open** subroutine is unsuccessful if any ofthe following conditions are true:

Not Applicable to PIOFS

- The file supports enforced record locks and another process has locked a portion of the file.
- The file is on a physical file system and is already open with the **O_RSHARE** flag or the **O_NSHARE** flag.

End of Not Applicable to PIOFS

- The file does not allow write access.

Not Applicable to PIOFS

- The file is already opened for deferred update.

End of Not Applicable to PIOFS

Not Applicable to PIOFS

OFlag Parameter Flag Values That Specify Type of Update

A program can request some type of update control on when updates should be made permanent for a regular file opened for write access. The following **OFlag** parameter value specifies the type of update performed:

O_SYNC If set, updates to regular files and writes to block devices that are synchronous updates. File update is performed by the following subroutines:

- **fclear**
- **ftruncate**
- **open** with O_TRUNC
- **write**

On return from a subroutine that performs a synchronous update (any of the preceding subroutines, when the **O_SYNC** flag is set), the program is assured that all data for the file has been writen to permanent storage, even if the file is also open for deferred update.

End of Not Applicable to PIOFS

Oflag FlagsThat Define the Open File Initial State

Not Applicable to PIOFS

The following **OFlag** parameter flag values define the initial state of the open file:

O_APPEND The file pointer is set to the end of the file prior to each write operation.

O_DELAY Specifies that if the **open** subroutine could not succeed due to an inability to grant the access on a physical file system required by the **O_RSHARE** flag or the **O_NSHARE** flag, the process blocks instead of returning the **ETXTBSY** error code.

O_NDELAY Opens with no delay.

O_NONBLOCK Specifies that the **open** subroutine should not block.

The **O_NDELAY** flag and the **O_NONBLOCK** flag are identical except for the value returned by the **read** and **write** subroutines. These flags mean the process does not block on the state of an object, but does block on input or output to a regular file or block device.

The **O_DELAY** flag is relevant only when used with the **O_NSHARE** or **O_RSHARE** flags. It is unrelated to the **O_NDELAY** and **O_NONBLOCK** flags.

General Notes on OFlag Parameter Flags

The effect of the **O_CREAT** flag is immediate, even if the file is opened with the **O_DEFER** flag.

When opening a file on a physical file system with the **O_NSHARE** flag or the **O_RSHARE** flag, if the file is already open with conflicting access the following can occur:

- If the **O_DELAY** flag is clear (the default), the open subroutine is unsuccessful.
- If the **O_DELAY** flag is set, the open subroutine blocks until there is no conflicting open. There is no deadlock detection for processes using the **O_DELAY** flag.

When opening a file on a physical file system that has already been opened with the **O_NSHARE** flag, the following can occur:

- If the **O_DELAY** flag is clear (the default), the open is unsuccessful immediately.
- If the **O_DELAY** flag is set, the open blocks until there is no conflicting open.

When opening a file with the **O_RDWR**, **O_WRONLY**, or **O_TRUNC** flag, and the file is already open with the **O_RSHARE** flag:

- If the **O_DELAY** flag is clear (the default), the open is unsuccessful immediately.
- If the **O_DELAY** flag is set, the open blocks until there is no conflicting open.

When opening a first-in-first-out (FIFO) with the **O_RDONLY** flag, the following can occur:

- If the **O_NDELAY** and **O_NONBLOCK** flags are clear, the open blocks until a process opens the file for writing. If the file is already open for writing (even by the calling process),he open subroutine returns without delay.
- If the **O_NDELAY** flag or the **O_NONBLOCK** flag is set, the open succeeds immediately even if no process has the FIFO open for writing.

When opening a FIFO with the **O_WRONLY** flag, the following can occur:

- If the **O_NDELAY** and **O_NONBLOCK** flags are clear (the default), the open blocks until a process opens the file for reading. If the file is already open for writing (even by the calling process) the **open** subroutine returns without delay.
- If the **O_NDELAY** flag or the **O_NONBLOCK** flag is set, the open subroutine returns an error if no process currently has the file open for reading.

When opening a block special or character special file that supports blocking opens, such as a terminal device, the following can occur:

- If the **O_NDELAY** and **O_NONBLOCK** flags are clear (the default), the open blocks until the device is ready or available.
- If the **O_NDELAY** flag or the **O_NONBLOCK** flag is set, the **open** subroutine returns without waiting for the device to be ready or available. Subsequent behavior of the device is device-specific.

Any additional information on the effect, if any, of the **O_NDELAY**, **O_RSHARE**, **O_NSHARE**, and **O_DELAY** flags on a specific device is documented in the description of the special file related to the device type.

End of Not Applicable to PIOFS

Return Values

Upon successful completion, the file descriptor, a nonnegative integer, is returned. Otherwise, a value of -1 is returned, no files are created or modified, and the **errno** global variable is set to indicate the error.

Error Codes

The **open**, **openx**, and **creat** subroutines are unsuccessful and the named file is not opened if one or more of the following are true:

EACCES One of the following is true:

- The file exists and the type of access specified by the *OFlag* parameter is denied.
- Search permission is denied on a component of the path prefix specified by the *Path* parameter. Access could be denied due to a secure mount.
- The file does not exist and write permission is denied for the parent directory of the file to be created.
- The **O_TRUNC** flag is specified and write permission is denied.

Not Applicable to PIOFS

EAGAIN The **O_TRUNC** flag is set and the named file contains a record lock owned by another process.

EDQUOT The directory in which the entry for the new link is being placed cannot be extended, or an inode could not be allocated for the file, because the user or group quota of disk blocks or inodes in the file system containing the directory has been exhausted.

End of Not Applicable to PIOFS

EEXIST The **O_CREAT** and **O_EXCL** flags are set and the named file exists.

EINTR A signal was caught during the **open** subroutine.

EISDIR Named file is a directory and write access is required (the **O_WRONLY** or **O_RDWR** flag is set in the *OFlag* parameter).

EMFILE The system limit for open file descriptors per process has already been reached (**OPEN_MAX**).

Not Applicable to PIOFS

ENAMETOOLONG The length of the *Path* parameter exceeds the system limit (**PATH_MAX**); or a path name component is longer than **NAME_MAX** and **_POSIX_NO_TRUNC** is in effect.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG The length of the *Path* parameter exceeded 255 characters or a path name component exceeded 255 characters.

End of Applicable only to PIOFS

ENFILE The system file table is full.

ENOENT The **O_CREAT** flag is not set and the named file does not exist; or the **O_CREAT** flag is not set and either the path prefix does not exist or the *Path* parameter points to an empty string.

ENOSPC The directory or file system that would contain the new file cannot be extended.

ENOTDIR A component of the path prefix specified by the *Path* component is not a directory.

Not Applicable to PIOFS

ENXIO One of the following is true:

- Named file is a character special or block special file, and the device associated with this special file does not exist.
- Named file is a multiplexed special file and either the channel number is outside of the valid range or no more channels are available.
- The **O_DELAY** flag or the **O_NONBLOCK** flag is set,the named file is a FIFO, the **O_WRONLY** flag is set, and no process has the file open for reading.

End of Not Applicable to PIOFS

EROFS Named file resides on a read-only file system and write access is required (either the **O_WRONLY**, **O_RDWR**, **O_CREAT** (if the file does not exist),or **O_TRUNC** flag is set in the *OFlag* parameter).

Not Applicable to PIOFS

ETXTBSY File is on a physical file system and is already open in a manner (with the **O_RSHARE** or **O_NSHARE flag**) that precludes this open;or the **O_NSHARE** or **O_RSHARE** flag was requested with the **O_NDELAY** flagset, and there is a conflicting open on a physical file system.

End of Not Applicable to PIOFS

The **open**, **openx**, and **creat** subroutines are unsuccessful if one of the following is true:

EFAULT The *Path* parameter points outside of the allocated address space of the process.

EINVAL The value of the *OFlag* parameter is not valid.

EINVAL A keyword has been specified incorrectly in the **piofs.profile** file.

Not Applicable to PIOFS

ELOOP Too many symbolic links were encountered in translating the *Path* parameter.

ETXTBSY The file specified by the *Path* parameter is a pure procedure (shared text) file that is currently executing, and the **O_WRONLY** or **O_RDWR** flag is set in the *OFlag* parameter.

End of Not Applicable to PIOFS

Related Information

Subroutines: **chmod**, **close**, **exec**, **fcntl**, **dup** or **dup2**, **fsync**, **ioctl**, **lockfx**, **lseek**, **read**, **stat**, **umask**, **write**.

putc, putchar, fputc, or putw

Purpose

Writes a character or a word to a stream.

Library

Standard I/O Package (**libc.a**)

C Syntax

```
#include <stdio.h>

int putc (
        int Character,
        FILE *Stream );

int putchar ( int Character );

int fputc (
        int Character,
        FILE *Stream );

int putw (
        int Word,
        FILE *Stream );
```

Parameters

Stream	Pointer to the file structure of an open file.
Character	A character to be written.
Word	A word to be written (not portable because word length and byte-ordering are machine dependent).

Description

The **putc** and **putchar** are macros used to write to a stream. The **fputc** and **putw** subroutines serve similar purposes but are true subroutines.

The **putc** macro writes the character *Character* to the output specified by the *Stream* parameter. The character is written at the position at which the file pointer is currently pointing, if defined.

The **putchar** macro is the same as the **putc** macro except that **putchar** writes to the standard output.

The **fputc** subroutine works the same as the **putc** macro, but **fputc** is a true subroutine rather than a macro. It runs more slowly than **putc**, but takes less space per invocation.

Because **putc** is implemented as a macro, it incorrectly treats a *Stream* parameter with side effects, such as **putc(***C***, ****f***++)**. For such cases, use the **fputc** subroutine instead. Also, use **fputc** whenever you need to pass a pointer to this subroutine as a parameter to another subroutine.

The **putc** and **putchar** macros have also been implemented as subroutines for ANSI compatibility. To access the subroutines instead of the macros, insert **#undef putc** or **#undef putchar** at the beginning of the source file.

The **putw** subroutine writes the word (**int**) specified by the *Word* parameter to the output specified by the *Stream* parameter. The word is written at the position at which the file pointer, if defined, is pointing. The size of a word is the size of an integer and varies from machine to machine. The **putw** subroutine does not assume or cause special alignment of the data in the file.

Because of possible differences in word length and byte ordering, files written using the **putw** subroutine are machine-dependent, and may not be readable using the **getw** subroutine on a different processor.

With the exception of **stderr**, output streams are, by default, buffered if they refer to files, or line-buffered if they refer to terminals. The standard error output stream, **stderr**, is unbuffered by default, but using the **freopen** subroutine causes it to become buffered or line-buffered. Use the **setbuf** subroutine to change the stream buffering strategy.

When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as it is written. When an output stream is buffered, many characters are saved and written as a block. When an output stream is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested).

Return Values

Upon successful completion, these functions each return the value written. If these functions fail, they return the constant EOF. They fail if the *Stream* parameter is not open for writing, or if the output file size cannot be increased. Because the EOF value is a valid integer, you should use the **ferror** subroutine to detect **putw** errors.

Error Codes

The **fputc** subroutine will fail if either the *Stream* is unbuffered or the *Stream* buffer needs to be flushed, and:

EAGAIN	The O_NONBLOCK flag is set for the file descriptor underlying *Stream* and the process would be delayed in the write operation.
EBADF	The file descriptor underlying *Stream* is not a valid file descriptor open for writing.

Not Applicable to PIOFS

EFBIG	An attempt was made to write a file that exceeds the file size of the process limit or the maximum file size.

End of Not Applicable to PIOFS

EINTR — The write operation was terminated due to the receipt of a signal, and either no data was transferred or the implementation does not report partial transfers for this file.

Note: Depending upon which library routine the application binds to, this subroutine may return EINTR. Refer to the **signal** subroutine regarding the **SA_RESTART** bit.

EIO — The process is a member of a background process group attempting to perform a **write** subroutine to its controlling terminal, the TOSTOP flag is set, the process is neither ignoring nor blocking the SIGTTOU signal and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.

ENOSPC — There was no free space remaining on the device containing the file.

Not Applicable to PIOFS

EPIPE — An attempt is made to write to a pipe or first-in-first-out (FIFO) that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

End of Not Applicable to PIOFS

The **fputc** subroutine may fail if:

ENOMEM — Insufficient storage space is available.

ENXIO — A request was made of a nonexistent device, or the request was outside the capabilities of the device.

Related Information

The **fclose** or **fflush** subroutine, **feof**, **ferror**, **clearerr**, or **fileno** subroutine, **fopen**, **freopen**, or **fdopen** subroutine, **fread** or **fwrite** subroutine, **getc**, **fgetc**, **getchar**, or **getw** subroutine, **getwc**, **fgetwc**, or **getwchar** subroutine, **printf**, **fprintf**, **sprintf**, **NLprintf**, **NLfprintf**, **NLsprintf**, or **sprintf** subroutine, **putwc**, **fputwc**, or **putwchar** subroutine, **puts** or **fputs** subroutine, **setbuf** subroutine.

puts or fputs

Purpose

Writes a string to a stream.

Library

Standard I/O Library (**libc.a**)

C Syntax

```
#include <stdio.h>

int puts ( const char *String );

int fputs (
        const char *String,
        FILE *Stream );
```

Parameters

String	*Pointer* to a string to be written to output.
Stream	*Pointer* to the FILE structure of an open file.

Description

The **puts** subroutine writes the string pointed to by the *String* parameter to the standard output stream, **stdout**, and appends a new-line character to the output.

The **fputs** subroutine writes the null-terminated string pointed to by the *String* parameter to the output stream specified by the *Stream* parameter. The **fputs** subroutine does not append a newline character.

Neither subroutine writes the terminating null character.

Return Values

Upon successful completion, the **puts** and **fputs** subroutines return the number of characters written. Otherwise, both subroutines return EOF, set an error indicator for the stream and set the **errno** global variable to indicate the error. This happens if the routines try to write on a file that has not been opened for writing.

Error Codes

The **puts** subroutine and the **fputs** subroutine are unsuccessful if the output stream specified by the *Stream* parameter is unbuffered or the buffer needs to be flushed and one or more of the following is true:

EAGAIN	The O_NONBLOCK flag is set for the file descriptor specified by the *Stream* parameter and the process would be delayed in the write operation.
EBADF	The file descriptor specified by the *Stream* parameter is not a valid file descriptor open for writing.

Not Applicable to PIOFS

EFBIG — An attempt was made to write to a file that exceeds the file size limit of the process or the systemwide maximum file size.

End of Not Applicable to PIOFS

EINTR — The write operation was terminated due to receipt of a signal and no data was transferred.

Note: Depending upon which library routine the application binds to, this subroutine may return EINTR. Refer to the **signal** subroutine regarding the **SA_RESTART** bit.

EIO — The process is a member of a background process group attempting to perform a write to its controlling terminal, the TOSTOP flag is set, the process is neither ignoring or blocking the SIGTTOU signal, and the process group of the process has no parent process.

ENOSPC — There was no free space remaining on the device containing the file specified by the *Stream* parameter.

Not Applicable to PIOFS

EPIPE — An attempt is made to write to a pipe or first-in-first-out (FIFO) that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

End of Not Applicable to PIOFS

ENOMEM — Insufficient storage space is available.

ENXIO — A request was made of a nonexistent device, or the request was outside the capabilities of the device.

Related Information

The **fopen**, **freopen**, or **fdopen** subroutine, **fread**, or **fwrite** subroutine, **gets** or **fgets** subroutine, **getws** or **fgetws** subroutine, **printf**, **fprintf**, and **sprintf** subroutine, **putws** or **fputws** subroutine, **putc**, **putchar**, **fputc**, or **putw** subroutine, **putwc**, **putwchar**, **fputwc** or **fputws** subroutine.

The **feof**, **ferror**, **clearerr**, or **fileno** macros.

read, readx, readv, or readvx

Purpose

Applicable only to PIOFS

read, or **readv** – Reads from a subfile.

End of Applicable only to PIOFS

Not Applicable to PIOFS

read, **readx**, **readv**, or **readvx** – Reads from a file.

End of Not Applicable to PIOFS

C Syntax

#include <**unistd.h**>

ssize_t read (*FileDescriptor, Buffer, NBytes***)**
int *FileDescriptor*;
void **Buffer*;
size_t *NBytes*;

Not Applicable to PIOFS

int readx (*FileDescriptor, Buffer, NBytes, Extension***)**
int *FileDescriptor*;
char **Buffer*;
unsigned int *NBytes*;
int *Extension*;

End of Not Applicable to PIOFS

#include <**sys/types.h**>
#include <**uio.h**>

int readv (*FileDescriptor, iov, iovCount***)**
int *FileDescriptor*;
struct iovec **iov*;
int *iovCount*;

Not Applicable to PIOFS

int readvx (*FileDescriptor, iov, iovCount*, *Extension***)**
int *FileDescriptor*;
struct iovec **iov*;
int *iovCount*;
int *Extension*;

End of Not Applicable to PIOFS

Description

The **read** subroutine attempts to read *NBytes* of data from the file associated with the *FileDescriptor* parameter into the buffer pointed to by the *Buffer* parameter.

The **readv** subroutine performs the same action but scatters the input data into the *iovCount* buffers specified by the array of **iovec** structures pointed to by the *iov* parameter. Each **iovec** entry specifies the base address and length of an area in memory where data should be placed. The **readv** subroutine always fills an area completely before proceeding to the next.

Not Applicable to PIOFS

The **readx** and **readvx** subroutines are the same as the **read** and **readv** subroutines, respectively, with the addition of an *Extension* parameter, which is needed when reading from some device drivers and when reading directories. While directories can be read directly, it is recommended that the **opendir** and **readdir** calls be used instead, as this is a more portable interface.

On regular files and devices capable of seeking...,

End of Not Applicable to PIOFS

the **read** starts at a position in the file given by the file pointer associated with the *FileDescriptor* parameter. Upon return from the **read** subroutine, the file pointer is incremented by the number of bytes actually read.

Not Applicable to PIOFS

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

On directories, the **readvx** subroutine starts at the position specified by the file pointer associated with the *FileDescriptor* parameter. The value of this file pointer must be either 0 or a value which the file pointer had immediately after a previous call to the **readvx** subroutine on this directory. Upon return from the **readvx** subroutine, the file pointer increments by a number that may not correspond to the number of bytes copied into the buffers.

When attempting to read from an empty pipe (first-in-first-out (FIFO)):

- If no process has the pipe open for writing, the **read** returns 0 to indicate end-of-file.
- If some process has the pipe open for writing:
  - If **O_NDELAY** and **O_NONBLOCK** are clear (the default), the **read** blocks until some data is written or the pipe is closed by all processes that had opened the pipe for writing.
  - If **O_NDELAY** is set, the **read** subroutine returns a value of 0.
  - If **O_NONBLOCK** is set, the **read** subroutine returns a value of -1 and sets the global variable **errno** to **EAGAIN.**

When attempting to read from a character special file that supports nonblocking reads, such as a terminal, and no data is currently available:

- If **O_NDELAY** and **O_NONBLOCK** are clear (the default), the **read** subroutine blocks until data becomes available.
- If **O_NDELAY** is set, the **read** subroutine returns 0.
- If **O_NONBLOCK** is set, the **read** subroutine returns -1 and sets the **errno** global variable to **EAGAIN** if no data is available.

When attempting to read a regular file that supports enforcement mode record locks, and all or part of the region to be read is currently locked by another process:

- If **O_NDELAY** and **O_NONBLOCK** are clear, the read blocks the calling ling process until the lock is released.
- If **O_NDELAY** or **O_NONBLOCK** is set, the read returns -1 and sets the global variable **errno** to **EAGAIN.**

End of Not Applicable to PIOFS

The behavior of an interrupted **read** subroutine depends on how the handler for the arriving signal was installed.

Not Applicable to PIOFS

Note: A read from a regular file is not interruptible. Only reads from objects that may block indefinitely, such as FIFOs, sockets, and some devices, are generally interruptible.

End of Not Applicable to PIOFS

Applicable only to PIOFS

Note: A read from a regular file is interruptible.

End of Applicable only to PIOFS

If the handler was installed with an indication that subroutines should not be restarted, the **read** subroutine returns a value of -1 and the global variable **errno** is set to **EINTR** (even if some data was already consumed).

If the handler was installed with an indication that subroutines should be restarted:

- If no data had been read when the interrupt was handled, this **read** will not return a value (it is restarted).
- If data had been read when the interrupt was handled, this **read** subroutine returns the amount of data consumed.

Parameters

FileDescriptor A file descriptor identifying the object to be read.

Extension Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the *Extension* parameter in a device-dependent way, either as a value or as a pointer to a communication area. Drivers must apply reasonable defaults when the value of the *Extension* parameter is 0.

For directories, the *Extension* parameter determines the format in which directory entries should be returned:

- If the value of the *Extension* parameter is 0, the format in which directory entries are returned depends on the value of the **real directory read** flag (described in the **ulimit** subroutine).
- If the calling process does not have the **real directory read** flag set, the buffers are filled with an array of directory en- tries truncated to fit the format of the System V directory structure. This provides compatibility with programs written for UNIX System V.
- If the calling process has the real directory read flag set (see the ulimit subroutine), the buffers are filled with an image of the underlying implementation of the directory.
- If the value of the *Extension* parameter is 1, the buffers are filled with consecutive directory entries in the format of a dirent structure. This is logically equivalent to the **readdir** subroutine.
- Other values of the *Extension* parameter are reserved.

For tape devices, the *Extension* parameter determines the response of the **readx** subroutine when the tape drive is in variable block mode and the read request is for less than the tape's block size.

- If the value of the *Extension* parameter is **TAPE_SHORT_READ,** the **readx** subroutine returns the number of bytes requested and sets the **errno** global variable to a value of 0.
- If the value of the *Extension* parameter is 0, the **readx** subroutine returns a value of 0 and sets the **errno** global variable to **ENOMEM.**

iov Points to an array of **iovec** structures that identifies the buffers into which the data is to be placed. The **iovec** structure is defined in the sys/uio.h file and contains the following members:

```
caddr_t iov_base;
int iov_len;
```

iovCount Specifies the number of **iovec** structures pointed to by the *iov* parameter.

Buffer Points to the buffer.

NBytes Specifies the number of bytes read from the file associated with the *FileDescriptor* parameter.

Not Applicable to PIOFS

Note: When reading tapes, the **read** subroutines consume a physical tape block on each call to the subroutine. If the physical data block size is larger than specified by the *Nbytes* parameter, an error will be returned, since all of the data from the **read** will not fit into the buffer specified by the read.

To avoid read errors due to unknown blocking sizes on tapes, set the *NBytes* parameter to a very large value (such as 32K bytes).

End of Not Applicable to PIOFS

Return Values

Upon successful completion, the **read, readx, readv,** and **readvx** subroutines return the number of bytes actually read and placed into buffers. The system guarantees to read the number of bytes requested if the descriptor references a normal file that has the same number of bytes left before the end of the file is reached, but in no other case.

A value of 0 is returned when the end of the file has been reached. (For information about communication files, see the **ioctl** and **termio** files.)

Otherwise, a value of -1 is returned, the global variable **errno** is set to identify the error, and the content of the buffer pointed to by the *Buffer* or *iov* parameter is indeterminate.

Error Codes

The **read, readx, readv,** and **readvx** subroutines are unsuccessful if one or more of the following are true:

EBADF The *FileDescriptor* parameter is not a valid file descriptor open for reading.

EINVAL The file position pointer associated with the *FileDescriptor* parameter was negative.

Applicable only to PIOFS

This is not checked if the Parallel I/O File System is using its internal 64-bit offset.

End of Applicable only to PIOFS

EINVAL The sum of the iov_len values in the *iov* array was negative or overflowed a 32-bit integer.

EINVAL The value of the *iovCount* parameter was not between 1 and 16, inclusive.

Not Applicable to PIOFS

EAGAIN The file was marked for non-blocking I/O, and no data was ready to be read.

End of Not Applicable to PIOFS

EFAULT The *Buffer* or part of the *iov* points to a location outside of the allocated address space of the process.

Not Applicable to PIOFS

EDEADLK A deadlock would occur if the calling process were to sleep until the region to be read was unlocked.

End of Not Applicable to PIOFS

EINTR A **read** was interrupted by a signal before any data arrived, and the signal handler was installed with an indication that subroutines are not to be restarted.

EIO An I/O error occurred while reading from the file system.

EIO The process is a member of a background process attempting to read from its controlling terminal, and either the process is ignoring or blocking the **SIGTTIN** signal or the process group has no parent process.

Not Applicable to PIOFS

The **read, readx, readv,** and **readvx** subroutines may be unsuccessful if the following is true:

ENXIO A request was made of a nonexistent device, or the request was outside the capabilities of the device.

If Network File System (NFS) is installed on the system, the **read** system call can also fail if the following is true:

End of Not Applicable to PIOFS

ETIMEDOUT The connection timed out.

Related Information

Subroutines: **open, openx, creat fcntl, dup, dup2 ioctl lockfx lseek opendir, readdir, seekdir, pipe, poll socket socketpair**

remove

Purpose

Removes a file.

Library

Standard C Library (**libc.a**)

C Syntax

#include <**stdio.h**>

int remove (char ******FileName* **);**

Parameters

FileName Specifies the name of the file being removed.

Description

The **remove** subroutine makes a file named by *FileName* inaccessible by that name. An attempt to open that file using that name does not work, unless you recreate it.

Not Applicable to PIOFS

If the file is open, it is not removed.

End of Not Applicable to PIOFS

Applicable only to PIOFS

You can remove a directory even if the directory is open.

End of Applicable only to PIOFS

If the file designated by the *FileName* parameter has multiple links, the link count of files linked to the removed file is reduced by 1.

Return Values

Upon successful completion, the **remove** subroutine returns a value of 0; otherwise it returns a nonzero value.

Related Information

The **link** subroutine, **rename** subroutine.

The **link** or **unlink** command.

rename

Purpose

Renames a directory or a file.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <stdio.h>

int rename (
        const char *FromPath,
        const char *ToPath );
```

Parameters

FromPath	Identifies the file or directory to be renamed.
ToPath	Identifies the new path name of the file or directory to be renamed. If *ToPath* is an existing file or empty directory, it is replaced by *FromPath*. If *ToPath* specifies a directory that is not empty, the **rename** subroutine exits with an error.

Description

The **rename** subroutine renames a directory or a file within a file system.

To use the **rename** subroutine, the calling process must have write and search permission in the parent directories of both the *FromPath* and *ToPath* parameters. If the path defined in the *FromPath* parameter is a directory, the calling process must have write and search permission to the *FromPath* directory as well.

If the *FromPath* and *ToPath* parameters both refer to the same existing file, the **rename** subroutine returns successfully and performs no other action.

The components of both the *FromPath* and *ToPath* parameters must be of the same type (that is, both directories or both non-directories) and must reside on the same file system. If the *ToPath* file already exists, it is first removed. Removing it guarantees that a link named *ToPath* will exist throughout the operation. This link refers to the file named by either the *ToPath* or *FromPath* parameter before the operation began.

If the final component of the *FromPath* parameter is a symbolic link, the symbolic link (not the file or directory to which it points) is renamed. If *ToPath* is a symbolic link, the link is destroyed.

If the parent directory of the *FromPath* parameter has the Sticky bit attribute (described in the **sys**/**mode.h** file), the calling process must have an effective user ID equal to the owner ID of the *FromPath* parameter, or to the owner ID of the parent directory of the *FromPath* parameter.

A user who is not the owner of the file or directory must have root user authority to use the **rename** subroutine.

If the *FromPath* and *ToPath* parameters name directories, the following must be true:

- The directory specified by the *FromPath* parameter is not an ancestor of *ToPath*. For example, the *FromPath* path name must not contain a path prefix that names the directory specified by the *ToPath* parameter.
- The directory specified in the *FromPath* parameter must be well-formed. A well-formed directory contains both . (dot) and .. (dot dot) entries. That is, the . (dot) entry in the *FromPath* directory refers to the same directory as that in the *FromPath* parameter. The .. (dot dot) entry in the *FromPath* directory refers to the directory that contains an entry for *FromPath*.
- The directory specified by the *ToPath* parameter, if it exists, must be well-formed (as defined previously).

Applicable only to PIOFS

- You can rename a directory even if the directory is open.
- You cannot rename an open file.

End of Applicable only to PIOFS

Return Values

Upon successful completion, the **rename** subroutine returns a value of 0. Otherwise, a value of -1 is returned, and the **errno** global variable is set to indicate the error.

Error Codes

The **rename** subroutine is unsuccessful and the file or directory name remains unchanged if one or more of the following are true:

EACCES	Creating the requested link requires writing in a directory mode that denies the process write permission.
EBUSY	The directory named by the *FromPath* or *ToPath* parameter is currently in use by the system.
EDQUOT	The directory that would contain the path specified by the *ToPath* parameter cannot be extended because the user's or group's quota of disk blocks on the file system containing the directory is exhausted.
EEXIST	The *ToPath* parameter specifies an existing directory that is not empty.
EINVAL	The path specified in the *FromPath* or *ToPath* parameter is not a well-formed directory (*FromPath* is an ancestor of *ToPath*), or an attempt has been made to rename . (dot) or .. (dot dot).
EISDIR	The *ToPath* parameter names a directory and the *FromPath* parameter names a non-directory.

EMLINK	The *FromPath* parameter names a directory that is larger than the maximum link count of the parent directory of the *ToPath* parameter.
ENOENT	A component of either path does not exist, the file named by the *FromPath* parameter does not exist, or a symbolic link was named, but the file to which it refers does not exist.
ENOSPC	The directory that would contain the path specified in the *ToPath* parameter cannot be extended because the file system is out of space.
ENOTDIR	The *FromPath* parameter names a directory and the *ToPath* parameter names a non-directory.
ENOTEMPTY	The *ToPath* parameter specifies an existing directory that is not empty.

Not Applicable to PIOFS

EROFS	The requested operation requires writing in a directory on a read-only file system.

End of Not Applicable to PIOFS

ETXTBSY	The *ToPath* parameter names a shared text file that is currently being used.
EXDEV	The link named by the *ToPath* parameter and the file named by the *FromPath* parameter are on different file systems. If Network File System (NFS) is installed on the system, the **rename** subroutine can be unsuccessful if the following is true:
ETIMEDOUT	The connection timed out.

The **rename** subroutine can be unsuccessful for other reasons. See “Error Codes for Services that Require Path Name Resolution” on page 125 for a list of additional errors.

Related Information

The **chmod** command, **mkdir** command, **mv** command, **mvdir** command.

The **chmod** subroutine, **link** subroutine, **mkdir** subroutine, **rmdir** subroutine, **unlink** subroutine.

rmdir

Purpose

rmdir – Removes a directory.

C Syntax

#include <**unistd.h**>

int rmdir (*Path***)**
const char ****Path*;**

Description

The **rmdir** subroutine removes the directory specified by the *Path* parameter. If Network File System (NFS) is installed on your system, this path can cross into another node.

For the **rmdir** subroutine to execute successfully, the calling process must have write access to the parent directory of the *Path* parameter.

In addition, if the parent directory of *Path* has the **Sticky** bit attribute (described in the **sys/mode.h** file), the calling process must have one of the following:

- An effective user ID equal to the directory to be removed
- An effective user ID equal to the owner ID of the parent directory of *Path*
- Root user authority.

Parameters

Path — Specifies the directory path name. The directory you specify must be:

Empty — The directory contains no entries other than . (dot) and .. (dot dot).

Well-formed — If the . (dot) entry in the *Path* parameter exists, it must refer to the same directory as *Path.* Exactly one directory has a link to the *Path* parameter, excluding the self-referential . (dot). If the .. (dot dot) entry in *Path* exists, it must refer to the directory that contains an entry for *Path.*

Return Values

Upon successful completion, the **rmdir** subroutine returns a value of 0. Otherwise, a value of -1 is returned, the specified directory is not changed, and the **errno** global variable is set to indicate the error.

Error Codes

The **rmdir** subroutine fails and the directory is not deleted if the following errors occur:

EACCES There is no search permission on a component of the path prefix, or there is no write permission on the parent directory of the directory to be removed.

EBUSY The directory is in use as a mount point.

EEXIST or ENOTEMPTY The directory named by the *Path* parameter is not empty.

Not Applicable to PIOFS

ENAMETOOLONG The length of the *Path* parameter exceeds **PATH_MAX;** or a path-name component longer than **NAME_MAX** and **POSIX_NO_TRUNC** is in effect.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG The length of the *Path* parameter exceeded 255 characters or a path-name component exceeded 255 characters.

End of Applicable only to PIOFS

ENOENT The directory named by the *Path* parameter does not exist, or the *Path* parameter points to an empty string.

ENOTDIR A component specified by the *Path* parameter is not a directory.

EINVAL The directory named by the *Path* parameter is not well-formed.

Not Applicable to PIOFS

EROFS The directory named by the *Path* parameter resides on a read-only file system.

End of Not Applicable to PIOFS

The **rmdir** subroutine can be unsuccessful for other reasons. See “Error Codes for Services that Require Path Name Resolution” on page 125 for a list of additional errors.

Not Applicable to PIOFS

If NFS is installed on the system, the **rmdir** subroutine fails if the following is true:

End of Not Applicable to PIOFS

ETIMEDOUT The connection timed out.

Related Information

Subroutines: **chmod**, **fchmod**, **mkdir**, **rename**, **remove**, **umask**, **unlink.**

Commands: **rm**, **rmdir.**

stat, lstat, fstat, statx, fstatx, fullstat, or ffullstat

Purpose

statx, **stat, fstatx, fstat, fullstat,** or **ffullstat** – Provides information about a file.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <sys/stat.h>

int stat (Path, Buffer)
char *Path;
struct stat *Buffer;

int lstat (Path, Buffer)
char *Path;
struct stat *Buffer;

int fstat (FileDescriptor, Buffer)
int FileDescriptor;
struct stat *Buffer;

int statx (Path, Buffer, Length, Command)
char *Path;
struct stat *Buffer;
int Length;
int Command;

int fstatx (FileDescriptor, Buffer, Length, Command)
int FileDescriptor;
struct stat *Buffer;
int Length;
int Command;

#include <sys/fullstat.h>

int fullstat (Path, Path, Command, Buffer)
struct fullstat *Buffer;
char *Path;
int Command;

int ffullstat (FileDescriptor, Command, Buffer)
struct fullstat *Buffer;
int FileDescriptor;
int Command;
```

stat, lstat, fstat, statx, fstatx, fullstat, or ffullstat

Description

The **stat** subroutine obtains information about the file named by the *Path* parameter. Read, write, or execute permission for the named file is not required, but all directories listed in the path leading to the file must be searchable. The file information, which is a subset of the **stat** structure, is written to the area specified by the *Buffer* parameter.

The **lstat** subroutine obtains information about a file that is a symbolic link, in which case **lstat** returns information about the link, while **stat** returns information about the file referenced by the link.

The **fstat** subroutine obtains information about the open file referenced by the *FileDescriptor* parameter. The **fstatx** subroutine obtains an open file referenced by the *FileDescriptor* parameter as in **fstat.**

The **statx** subroutine obtains a greater set of file information than the **stat** subroutine. The *Path* parameter is processed diffferently, depending on the contents of the *Command* parameter. The *Command* parameter provides the ability to collect information about symbolic links (as with the **lstat** subroutine) as well as information about mount points and hidden directories. The **statx** subroutine returns the amount of information specified by the *Length* parameter.

The **fullstat** and **ffullstat** subroutines are interfaces maintained for backward compatibility With the exception of some field names, the **fullstat** structure is identical to the **stat** structure.

Parameters

Path — The path name identifying the file. This name can be interpreted differently depending on the interface used.

FileDescriptor — The file descriptor identifying the open file.

Buffer — A pointer to the **stat** structure in which information is returned. The **stat** structure is described in the **sys/stat.h** file.

Length — Indicates the amount of information, in bytes, to be returned. Any value between 0 and the STATXSIZE macro may be specified. The following macros may be used:

- **STATSIZE** — The subset of the **stat** structure that is normally returned for a **stat** call.
- **FULLSTATSIZE** — The subset of the **stat** (fullstat) structure that is normally returned for a fullstat call.
- **STATXSIZE** — The complete **stat** structure. A *Length* value of 0 is equivalent to the STATXSIZE macro.

Command — Specifies one of the following processing options.

For the **statx** subroutine, the *Command* parameter determines how to interpret the path name provided; specifically, whether to retrieve information about a symbolic link, hidden directory, or mount point. Flags can be combined by logically ORing them together.

STX_LINK If the *Command* parameter specifies the STX_LINK flag and the *Path* parameter is a path name that refers to a symbolic link, **statx** returns information about the symbolic link. If the **STX_LINK** flag is not specified, **statx** returns information about the file to which the link refers.

If *Command* specifies the **STX_LINK** flag and the *Path* value refers to a symbolic link, the **st_mode** field of the returned **stat** structure indicates that the file is a symbolic link.

STX_HIDDEN If *Command* specifies the **STX_HIDDEN** flag and the *Path* value is a path name that refers to a hidden directory, **statx** returns information about the hidden directory. If the **STX_HIDDEN** flag is not specified, **statx** returns information about a subdirectory of the hidden directory.

If *Command* specifies the **STX_HIDDEN** flag and *Path* refers to a hidden directory, the **st_mode** field of the returned **stat** structure indicates that this is a hidden directory.

STX_MOUNT If *Command* specifies the **STX_MOUNT** flag and the *Path* value is the name of a file or directory that has been mounted over, **statx** returns information about the mounted-over file. If the **STX_MOUNT** flag is not specified, **statx** returns information about the mounted file or directory (the root of a virtual file system).

If *Command* specifies the **STX_MOUNT** flag, the **FS_MOUNT** bit in the **st_flag** field of the returned **stat** structure is set if (and only if) this file is mounted over.

If *Command* does not specify the **STX_MOUNT** flag, the **FS_MOUNT** bit in the **st_flag** field of the returned **stat** structure is set if (and only if) this file is the root of a virtual file system.

STX_NORMAL If *Command* specifies the **STX_NORMAL** flag, then no special processing is performed on the *Path* value. This option should be used when **STX_LINK, STX_HIDDEN,** and **STX_MOUNT** flags are not desired.

For the **fstatx** subroutine, there are currently no special pro- cessing options. The only valid value for *Command* is the **STX_NORMAL** flag.

For **fullstat** and **ffullstat,** *Command* may be the **FL_STAT** flag, which is equivalent to the **STX_NORMAL** flag, or the **FL_NOFOLLOW** flag, which is equivalent to **STX_LINK** flag.

Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **stat** subroutine is unsuccessful if one or more of the following are true:

EACCES Search permission is denied for one component of the path prefix.

Not Applicable to PIOFS

ENAMETOOLONG The length of the path prefix exceeds the **PATH_MAX** flag value or a path name is longer than the **NAME_MAX** flag value while the **POSIX_NO_TRUNC** flag is in effect.

End of Not Applicable to PIOFS

Applicable only to PIOFS

ENAMETOOLONG The length of the path prefix exceeded 255 characters or a path name exceeded 255 characters.

End of Applicable only to PIOFS

ENOTDIR A component of the path prefix is not a directory.

The **stat,** lstat, **statx,** and **fullstat** subroutines are unsuccessful if one or more of the following are true:

ENOENT The file named by *Path* does not exist.

EFAULT Either the *Path* or the *Buffer* parameter points to a location outside of the allocated address space of the process.

ENOENT The file named by *Path* does not exist.

The **stat,** lstat, **statx,** and **fullstat** subroutines can be unsuccessful for other reasons. See “Error Codes for Services that Require Path Name Resolution” on page 125 for a list of additional errors.

The **fstat, fstatx,** and **ffullstat** subroutines fail if one or more of the following are true:

EBADF *FileDescriptor* is not a valid file descriptor.

EFAULT The *Buffer* parameter points to a location outside of the allocated address space of the process.

EIO An I/O error occurred while reading from the file system.

The **statx** and **fstatx** subroutines are unsuccessful if one or more of the following are true:

EINVAL The *Length* value is not between 0 and the STATSIZE macro.

EINVAL An illegal value was provided for the *Command* parameter.

Related Information

Commands: **chmod, chown, link, mknod, mount, openx, open,** or **creat, pipe, symlink, vtimes.**

unlink

Purpose

unlink – Removes a directory entry.

Library

Standard C Library (**libc.a**)

C Syntax

```
#include <unistd.h>

int unlink (Path)
char *Path;
```

Description

The **unlink** subroutine removes the directory entry specified by the *Path* parameter and decreases the link count of the file referenced by the link. If Network File System (NFS) is installed on your system, this path can cross into another node.

Warning: Removing a link to a directory requires root user authority. Unlinking of directories is strongly discouraged since erroneous directory structures can result. The **rmdir** subroutine should be used to remove empty directories.

Not Applicable to PIOFS

When all links to a file are removed and no process has the file open, all resources associated with the file are reclaimed, and the file is no longer accessible. If one or more processes have the file open when the last link is removed, the directory entry disappears. However, the removal of the file contents is postponed until all references to the file are closed.

If the parent directory of *Path* has the sticky attribute (described in the **mode.h** file), the calling process must have root user authority or an effective user ID equal to the owner ID of *Path* or the owner ID of the parent directory of *Path*.

End of Not Applicable to PIOFS

Applicable only to PIOFS

A file will be unlinked only if no processes are accessing the file.

End of Applicable only to PIOFS

The **st_ctime** and **st_mtime** fields of the parent directory are marked for update if the **unlink** subroutine is successful. In addition, if the file's link count is not 0, the **st_ctime** field of the file will be marked for update.

Applications should use the **rmdir** subroutine to remove a directory. If the *Path* parameter names a symbolic link, the link itself is removed.

Parameters

Path Specifies the directory entry to be removed.

Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, the **errno** global variable is set to indicate the error, and the specified file is not changed.

Error Codes

The **unlink** subroutine fails and the named file is not **unlinked** if one of the following is true:

ENOENT The named file does not exist.

EACCES Write permission is denied on the directory containing the link to be removed.

EPERM The named file is a directory, and the calling process does not have root user authority.

EBUSY The entry to be **unlinked** is the mount point for a mounted file system.

EPERM The file specified by the *Path* parameter is a directory, and the calling process does not have root user authority.

Not Applicable to PIOFS

EROFS The entry to be unlinked is part of a read-only file system.

The **unlink** subroutine can be unsuccessful for other reasons. For a list of additional errors, see “Error Codes for Services that Require Path Name Resolution” on page 125.

If NFS is installed on the system, the **unlink** subroutine can also fail if the following is true:

End of Not Applicable to PIOFS

ETIMEDOUT The connection timed out.

Related Information

Command: **rm**.

Subroutines: **close**, **open**, **remove**, **rmdir**.

write, writex, writev, or writevx

Purpose

Applicable only to PIOFS

write, or **writev** – Writes to a subfile.

End of Applicable only to PIOFS

Not Applicable to PIOFS

write, **writex**, **writev**, or **writevx** – Writes to a file.

End of Not Applicable to PIOFS

#include <**unistd.h**>

int write (*FileDescriptor, Buffer, NBytes***)**
int *FileDescriptor*;
const void **Buffer*;
size_t *NBytes*;

Not Applicable to PIOFS

int writex (*FileDescriptor, Buffer, NBytes, Extension***)**
int *FileDescriptor*;
char **Buffer*;
unsigned int *NBytes*;
int *Extension*;

End of Not Applicable to PIOFS

#include <**sys/types.h**>
#include <**uio.h**>

int writev (*FileDescriptor, iov, iovCount***)**
int *FileDescriptor*;
struct iovec **iov*;
int *iovCount*;

Not Applicable to PIOFS

int writevx (*FileDescriptor, iov, iovCount*, *Extension***)**
int *FileDescriptor*;
struct iovec **iov*;
int *iovCount*;
int *Extension*;

End of Not Applicable to PIOFS

Description

The **write** subroutine attempts to write the number of bytes of data specified by the *NBytes* parameter to the file associated with the *FileDescriptor* parameter from the buffer pointed to by the *Buffer* parameter.

The **writev** subroutine performs the same action but gathers the output data from the *iovCount* buffers specified by the array of **iovec** structures pointed to by the *iov* parameter. Each **iovec** entry specifies the base address and length of an area in memory from which data should be written. The **writev** subroutine always writes a complete area before proceeding to the next.

Not Applicable to PIOFS

The **writex** and **writevx** subroutines are the same as the **write** and **writev** subroutines, respectively, with the addition of an *Extension* parameter, which is used when writing to some device drivers.

With regular files and devices capable of seeking...

End of Not Applicable to PIOFS

...the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from the **write** subroutine, the file pointer increments by the number of bytes actually written.

Applicable only to PIOFS

With PIOFS, the file pointer is updated internally prior to the actual write. Thus, if the write returns with anything other than success, the file pointer may point beyond where the data has been written correctly.

End of Applicable only to PIOFS

Not Applicable to PIOFS

With devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

End of Not Applicable to PIOFS

Fewer bytes can be written than requested if there is not enough room to satisfy the request. In this case the number of bytes written is returned. The next attempt to write a nonzero number of bytes is unsuccessful (except as noted in the following text). The limit reached can be either that set by the **ulimit** subroutine or the end of the physical medium.

Successful completion of a **write** subroutine clears the **SetUserID** bit (**S_ISUID)** of a file if all of the following are true:

- The calling process does not have root user authority.

- The effective user ID of the calling process does not match the user ID of the file.
- The file is executable by the group (**S_IXGRP)** or other (**S_IXOTH).**

The **write** subroutine clears the **SetGroupID** bit (**S_ISGID**) if all of the following are true:

- The calling process does not have root user authority.
- The group ID of the file does not match the effective group ID or one of the supplementary group IDs of the process.
- The file is executable by the owner (**S_IXUSR**) or others (**S_IXOTH**).

Note: Clearing of the **SetUserID** and **SetGroupID** bits can occur even if the **write** subroutine is unsuccessful, if file data was modified before the error was detected.

If the **O_APPEND** flag of the file status is set, the file offset is set to the end of the file prior to each write.

Applicable only to PIOFS

O_APPEND is only valid for files opened as a single subfile.

End of Applicable only to PIOFS

Not Applicable to PIOFS

If the *FileDescriptor* parameter refers to a regular file whose file status flags specify **O_SYNC,** this is a synchronous update (as described in the **open** subroutine).

If the *FileDescriptor* parameter refers to a regular file that a process has opened with the **O_DEFER** file status flag set, the data and file size are not updated on permanent storage until a process issues an **fsync** subroutine or performs a synchronous update. If all processes that have the file open with the **O_DEFER** file status flag set close the file before a process issues an **fsync** subroutine or performs a synchronous update, the data and file size are not updated on permanent storage.

Write requests to a pipe (or first-in-first-out (FIFO)) are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe; hence, each write request appends to the end of the pipe.
- If the size of the **write** request is less than or equal to the value of the **PIPE_BUF** system variable (described in the **pathconf** routine), the **write** subroutine is guaranteed to be atomic. The data is not interleaved with data from other write processes on the same pipe. Writes of greater than **PIPE_BUF** bytes can have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the **O_NDELAY** or **O_NONBLOCK** file status flags are set.
- If the **O_NDELAY** and **O_NONBLOCK**

file status flags are clear (the default), a write request to a full pipe causes the process to block until enough space becomes available to handle the entire request.

- If the **O_NDELAY** file status flag is set, a write to a full pipe returns a 0.
- If the **O_NONBLOCK** file status flag is set, a write to a full pipe returns a value of -1 and sets the **errno** global variable to **EAGAIN.**

When attempting to write to a character special file that supports nonblocking writes, such as a terminal, and no data can currently be written:

- If the O_NDELAY and O_NONBLOCK flags are clear (the default), the **write** subroutine blocks until data can be written.
- If the O_NDELAY flag is set, the **write** subroutine returns 0.
- If the O_NONBLOCK flag is set, the **write** subroutine returns -1 and sets the **errno** global variable to EAGAIN if no data can be written.

When attempting to write to a regular file that supports enforcement-mode record locks, and all or part of the region to be written is currently locked by another process, the following can occur:

- If the **O_NDELAY** and **O_NONBLOCK** file status flags are clear (the default), the calling process blocks until the lock is released.
- If the **O_NDELAY** or **O_NONBLOCK** file status flag is set, then the **write** subroutine returns a value of -1 and sets the **errno** global variable to [EAGAIN].

Note: The **fcntl** subroutine provides more information about record locks.

End of Not Applicable to PIOFS

The behavior of an interrupted **write** subroutine depends on how the handler for the arriving signal was installed. The handler can be installed in one of two ways, with the following results:

- If the handler was installed with an indication that subroutines should not be restarted, the **write** subroutine returns a value of -1 and sets the **errno** global variable to **EINTR** (even if some data was already written).
- If the handler was installed with an indication that subroutines should be restarted, and:
  - If no data had been written when the interrupt was handled, the **write** subroutine will not return a value (it is restarted).
  - If data had been written when the interrupt was handled, this **write** subroutine returns the amount of data already written.

Not Applicable to PIOFS

Note: A write to a regular file is not interruptible. Only writes to objects that may block indefinitely, such as FIFOs, sockets, and some devices, are generally interruptible.

End of Not Applicable to PIOFS

Applicable only to PIOFS

A write to a regular file is interruptible.

End of Applicable only to PIOFS

Parameters

Buffer Identifies the buffer containing the data to be written.

Extension Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the *Extension* parameter in a device-dependent way, either as a value or as a pointer to a communication area. Drivers must apply reasonable defaults when the Extension parameter value is 0.

FileDescriptor Identifies the object to which the data is to be written.

iov Points to an array of **iovec** structures, which identifies the buffers containing the data to be written. The **iovec** structure is defined in the **sys/uio.h** file and contains the following members:

```
caddr_t iov_base;
int   iov_len;
```

iovCount Specifies the number of **iovec** structures pointed to by the *iov* parameter.

NBytes Specifies the number of bytes to write.

Return Values

Upon successful completion, the **write, writex, writev,** and **writevx** subroutines return the number of bytes that were actually written. The number of bytes written is never greater than the value specified by the *NBytes* parameter. Otherwise, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **write, writex, writev,** and **writevx** subroutines are unsuccessful when one of the following is true:

EBADF The *FileDescriptor* parameter does not specify a valid file descriptor open for writing.

Not Applicable to PIOFS

EDQUOT New disk blocks cannot be allocated for the file because the user or group quota of disk blocks has been exhausted on the file system.

End of Not Applicable to PIOFS

EINVAL The file position pointer associated with the *FileDescriptor* parameter was negative; the *iovCount* parameter value was not between 1 and 16, inclusive; or one of the iov_len values in the *iov* array was negative or the sum overflowed a 32-bit integer.

EFAULT The *Buffer* parameter or part of the *iov* parameter points to a location outside of the allocated address space of the process.

Not Applicable to PIOFS

EPIPE An attempt was made to **write** to a file that is not opened for reading by any process, or to a socket of type **SOCK_STREAM** that is not connected to a peer socket; or an attempt was made to write to a pipe or FIFO that is not open for reading by any process. If this occurs, a **SIGPIPE** signal will also be sent to the process.

EAGAIN The O_NONBLOCK flag is set on this file and the process would be delayed in the write operation; or an enforcement-mode record lock is outstanding in the portion of the file that is to be written.

EFBIG An attempt was made to write a file that exceeds the maximum file size.

End of Not Applicable to PIOFS

ENOSPC No free space is left on the file system containing the file.

EINTR A signal was caught during the write operation, and the signal handler was installed with an indication that subroutines are not to be restarted.

EIO An I/O error occurred while writing to the file system; or the process is a member of a background process group attempting to write to its controlling terminal, **TOSTOP** is set, the process is neither ignoring nor blocking **SIGTTOU,** and the process group has no parent process.

Not Applicable to PIOFS

The **write, writex, writev,** and **writevx** subroutines may be unsuccessful if the following is true:

ENXIO A request was made of a nonexistent device, or the request was outside the capabilities of the device.

End of Not Applicable to PIOFS

Related Information

open, openx, creat, fcntl, dup, dup2, fsync, ioctl, lockfx, lseek, pathconf, pipe, poll, select, ulimit.

Appendixes

© Copyright IBM Corp. 1995

Appendix A. Parallel I/O File System Messages

INFO1 **Usage: piofs_config [-u]** *config_file_name*

Explanation: **piofs_config** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* has a fully qualified path name.

INFO2 **Usage: piofs_lconfig [-u]** *ipaddr config_file_name*

Explanation: **piofs_lconfig** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* has a fully qualified pathname.

INFO3 *function_name*: **Hostname = (** *hostname* **) successfully configured.**

Explanation: The specified hostname was configured successfully.

User Response: None.

INFO4 *function_name*: **Hostname = (** *hostname* **) successfully unconfigured.**

Explanation: The specified hostname was unconfigured successfully.

User Response: None.

INFO5 *function_name*: **All nodes configured successfully.**

Explanation: All the nodes have been configured successfully.

User Response: None.

INFO6 *function_name*: **Unconfiguration is complete.**

Explanation: All nodes have been unconfigured succesfully.

User Response: None.

INFO07 *function_name*: **Warning, all data associated with /piofsdev1 through /piofsdev128 and /piofsmdata on all the specified server nodes may be lost. Enter 'yes' to continue, anything else to terminate.**

Explanation: The Parallel I/O File System is about to be configured or unconfigured.

User Response: Enter 'yes' if you wish to configure or unconfigure the Parallel I/O File System.

INFO8 **export LIBPATH=***nnnnnn***:/usr/lib:/lib to access > 2G PIOFS files**

Explanation: Export LIBPATH environment variable to permit unmodified programs calling **llseek()** to access greater than 2GB files using PIOFS

User Response: None.

INFO9 **Usage:** *nnnnnn* **XCOFF_file_path**

Explanation: Incorrect number of arguments.

User Response: Specify the correct number of arguments.

© Copyright IBM Corp. 1995

INFO200 Usage: *nnnnnn* **{ M** | **U } 0 nodename object stub options**

Explanation: **piofsmnthelp** was invoked incorrectly. This program should only be invoked by the AIX **mount** and **unmount** commands.

User Response: Collect as much information as you can about what was happening when this error occurred and follow your local procedures for getting help.

INFO201 Usage: installpiofs [-a | **-d** | **-q]**

Explanation: **installpiofs** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* has a fully qualified path name.

INFO202 *function_name***: The PIOFS kernel extension is loaded at** *nnn*

Explanation: The Parallel I/O File System kernel extension has been loaded.

User Response: None.

INFO203 *function_name***: The PIOFS kernel extension is not loaded.**

Explanation: The Parallel I/O File System kernel extension has not been loaded.

User Response: None.

INFO204 Usage: NetworkInterfaceNameToDottedIPaddress *network_interface_name*

Explanation: The NetworkInterfaceNameToDottedIPaddress *network_interface_name* was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO205 Usage: add_piofs_user *name mount_point* **[** *block_limit object_limit* **]**

Explanation: **add_piofs_user** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO206 Usage: delete_piofs_user *name mount_point*

Explanation: **delete_piofs_user** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO207 Usage: stat_piofs_user *name mount_point*

Explanation: **stat_piofs_user** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO208 uid=*nnn* **block_limit=***nnn* **blocks_used=***nnn* **object_limit=***nnn* **files=***nnn* **dirs=***nnn* **cells=***nnn*

Explanation: This is the information displayed in response to the **stat_piofs_user** command (AIX Version 4.1).

User Response: None.

INFO209 **uid=***nnn* **block_limit=***nnn* **blocks_used=***nnn* **object_limit=***nnn* **files=***nnn* **dirs=***nnn* **cells=***nnn*

Explanation: This is the information displayed in response to the **stat_piofs_user** command (AIX Version 3.2.5).

User Response: None.

INFO210 **Usage: list_piofs_users** *mount_point*

Explanation: **list_piofs_users** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO211 **uid=***nnn* **User_name=***nnn* **block_limit=***nnn* **object_limit=***nnn*

Explanation: This is the information displayed in response to the **list_piofs_users** command.

User Response: None.

INFO212 **Usage: change_piofs_user** *name mount_point block_limit object_limit*

Explanation: **change_piofs_user** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO213 **Usage: stat_piofs_server [** **–l]** *mount_point*

Explanation: **stat_piofs_server** was invoked incorrectly.

User Response: Verify the syntax and reissue the command.

INFO214 **status=***nnn* **server=***nnn* **host=***nnn* **disks=***nnn*

Explanation: This is the short form of the information displayed for each server node in response to the **stat_piofs_server** command.

User Response: None.

INFO215 **blocks(used/total)=***nnn*/*nnn* **(** *nnn***%)**
metadata=(*nnn***%) files=***nnn* **dirs=***nnn* **cells=***nnn*

Explanation: This is the additional information that is displayed for each server node in response to the **stat_piofs_server** command when the long form is requested.

User Response: None.

INFO400 **Usage: format_piofs_server** *config_file_name*

Explanation: **format_piofs_server** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* is **not** a fully qualified pathname.

INFO401 **Usage: start_piofs_server** *config_file_name*

Explanation: **start_piofs_server** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* is **not** a fully qualified pathname.

INFO402 Usage: stop_piofs_server *config_file_name*

Explanation: **stop_piofs_server** was invoked incorrectly.

User Response: Verify the arguments and ensure that the *config_file_name* is **not** a fully qualified pathname.

INFO403 Usage: *nnnnnn* **nameof.piofscfg**

Explanation: PiofsServer was invoked incorrectly.

User Response: Verify the arguments.

INFO404 Usage: *nnnnnn* **nameof.piofscfg [Format]**

Explanation: PiofsServerFormat was invoked incorrectly.

User Response: Verify the arguments.

INFO405 *function_name*: **Warning, about to format a PIOFS file system.**
All data will be lost.
Enter `Format` **to continue, anything else to terminate.**

Explanation: The Parallel I/O File System file system is about to be formatted.

User Response: Make sure you want to format the Parallel I/O File System files. You must enter the word `Format` with an upper case F.

INFO406 *function_name*: **Format program terminating.**

Explanation: Format program is stopping without formatting because you requested that it stop.

User Response: None.

INFO407 *function_name*: **The server has been started on** *nnnnnn* **node(s).**

Explanation: The server has been started.

User Response: None.

INFO408 *function_name*: **The server has been formatted on** *nnnnnn* **node(s).**

Explanation: The server has been formatted.

User Response: None.

INFO409 *function_name*: **Enter the hostname and X Server display number where the aixterm command is to run.**

Explanation: A hostname and X Server display number is needed for the aixterm command.

User Response: Enter the requested data.

INFO410 *function_name*: **Warning messages were issued while installing the pmp kernel extension, see** *file_name*.

Explanation: Warning messages were issued while installing the pmp kernel extension. See the specified log file.

User Response: The network options specified differed from those recommended in the README file for the Parallel I/O File System.

If you intended to use network options other than those recommended, disregard this message.

If you intended to specify the recommended options, remove the pmp kernel extension from the kernel, modify the network options, and re-install the pmp kernel extension.

INFO600 Usage: installpmp [-ai ipaddr | -d | -q]
[-p msg_port] [-q ack_port]
[-r udp_recvspace] [-s udp_sendspace]
[-t] [-c *config_file_name* **]**

Explanation: The **installpmp** command was invoked incorrectly.

User Response: Verify the arguments.

INFO601 *function_name***: The pmp kernel extension is loaded at 0x***nnn*

Explanation: The pmp kernel extension has been loaded.

User Response: None.

INFO602 *function_name***: The pmp kernel extension is not loaded.**

Explanation: The pmp kernel extension has not been loaded.

User Response: None.

INFO603 *function_name***: Warning from node** *node***: network option** *option***=** *nnn* **is below the suggested value of** *nnn***.**

Explanation: The pmp kernel extension has been installed with options that might cause problems.

User Response: If you intended to use network options other than those recommended, disregard this message.

If you intended to specify the recommended options, remove the pmp kernel extension from the kernel, modify the network options, and re-install the pmp kernel extension.

INFO604 *function_name***: Node** *node***: udp_sendspace =** *nnn***, udp_recvspace =** *nnn***; can effectively receive from** *nnn* **nodes**

Explanation: The **udp_sendspace** and **udp_recvspace** values used to initialize the pmp kernel extension will support the receiving of messages from the specified number of nodes.

User Response: None.

INFO605 *function_name***: Warning messages were issued from node** *node* **while installing the pmp kernel extension, see** *file_name*

Explanation: Warning messages were issued while installing the pmp kernel extension. See the specified log file.

User Response: If you intended to use network options other than those recommended, disregard this message.

If you intended to specify the recommended options, remove the pmp kernel extension from the kernel, modify the network options, and re-install the pmp kernel extension.

0024-001 *function_name***: Configuration statement in error:** *nnnnnn nnnnnn nnnnnn nnnnnn* **.**

Explanation: This configuration statement is in error.

User Response: Correct the error.

0024-002 *nnnnnn*: **Unable to allocate storage.**

Explanation: The system was unable to allocate storage. This is a system problem.

User Response: Contact your system support personnel.

0024-003 *function_name*: **Device numbers must be sequential starting at 1, with a maximum of 128.**

Explanation: Device numbers were specified incorrectly in the configuration file.

User Response: Fix the statement in error.

0024-004 *function_name*: **The logical storage node numbers must be sequential starting at 0.**

Explanation: Storage node numbers were specified incorrectly in the configuration file.

User Response: Fix the statement in error.

0024-005 *function_name*: **The IP addresses must all be the same for logical storage node (** *nnnnnn* **).**

Explanation: IP addresses were specified incorrectly in the configuration file.

User Response: Fix the statement in error.

0024-006 *function_name*: **The hostname in the configuration file (** *config_file_name* **) is incorrect. The hostname returned by the machine is (** *hostname* **).**

Explanation: The hostname specified in the configuration file is incorrect.

User Response: Fix the statement in error.

0024-007 *function_name*: **rsh to** *node* **failed.**

Explanation: The root user was unable to **rsh** to the specified node.

User Response: Set up your environment to allow the root user to **rsh** to all the server nodes.

0024-008 *function_name*: **Could not communicate with node** *node*; **continuing to configure the other nodes.**

Explanation: The root user was unable to **ping** the specified node.

User Response: Make sure that the specified node is running and that you can communicate with it over the IP address that is in the configuration file.

0024-009 *function_name*: **Errors configuring.**

Explanation: Errors were found during the configuration process.

User Response: See the messages following this message.

0024-010 *function_name*: **Could not communicate with node** *node* **to unconfigure it. Unconfigure it manually.**

Explanation: The root user was unable to **ping** the specified node.

User Response: Make sure that the specified node is running and that you can communicate with it over the IP address that is in the configuration file.

0024-011 *function_name*: **Metadata statement missing for hostname (** *hostname* **).**

Explanation: Each storage node specified in the configuration file must have one and only one metadata statement.

User Response: Add the missing statement.

0024-012 More than one metadata record was specified for hostname (*hostname* **).**

Explanation: Each storage node specified in the configuration file must have one and only one metadata statement.

User Response: Remove one of the metadata statements.

0024-013 *function_name*: **Hostname = (** *hostname* **) failed configuration.**

Explanation: The specified hostname failed configuration.

User Response: After **piofs_config,** completes, check the error log.

0024-014 *function_name*: **There must be at least one device statement for hostname (** *hostname* **).**

Explanation: Each storage node specified in the configuration file must have at least one device statement.

User Response: Correct the configuration file.

0024-015 *function_name*: **PIOFS does not appear to be installed on node** *node*.

Explanation: A PIOFS executable could not be in /**usr**/**lpp**/**piofs**/**bin** on the specified node.

User Response: Install PIOFS on the node or correct the configuration file.

0024-016 *function_name*: **Hostname = (** *hostname: directory* **) failed unconfiguration.**

Explanation: The specified hostname could not be unconfigured. This probably is due to a hardware error.

User Response: After the shell script, **piofs_config** completes, check the system error log.

0024-017 *function_name*: **Error remotely copying** *file_name* **to** *directory*.

Explanation: The root user was unable to **rcp** the specified file.

User Response: Make sure that root can **rsh** to all the storage nodes without supplying a password.

0024-018 *function_name*: **There was no active configuration to unconfigure.**

Explanation: A **piofs_config -u** command was issued, but there was no active configuration to unconfigure.

User Response: Check the name of the configuration file that was specified with **piofs_config**.

0024-019 *function_name*: **Error retrieving file** *config_file_name* **from the system data repository.**

Explanation: An error occured while trying to get the configuration file from the system data repository.

User Response: Check that the system data repository is functioning properly.

0024-020 *function_name*: **Only one block_limit statement is permitted.**

Explanation: Only one block_limit statement is permitted in the configuration file.

User Response: Remove the extra block_limit statements.

0024-021 *function_name*: **The block_limit must be a number between 1 and 134217727.**

Explanation: A block_limit was specified that is out of the acceptable range.

User Response: Correct the block_limit statement.

0024-022 *function_name*: **Only one object_limit statement is permitted.**

Explanation: Only one object_limit statement is permitted in the configuration file.

User Response: Remove the extra object_limit statements.

0024-023 *function_name*: **The object_limit must be a number between 1 and 13421772.**

Explanation: A object_limit was specified that is out of the acceptable range.

User Response: Correct the object_limit statement.

0024-024 *function_name*: **Only one IPport statement is permitted.**

Explanation: Only one IPport statement is permitted in the configuration file.

User Response: Remove the extra IPport statements.

0024-025 *function_name*: **Metadata statements must have a logical size of** *nnnnnn* **Megabytes.**

Explanation: The logical volumes built to contain metadata must be the specified size.

User Response: Correct the statement.

0024-026 *function_name*: **An invalid keyword** *keyword* **was specified in the configuration file.**

Explanation: An invalid keyword was specified in the configuration file.

User Response: Correct the statement.

0024-027 *function_name*: **No storage_node statements were found in the configuration file.**

Explanation: No storage_node statements were found in the configuration file.

User Response: Add the appropriate storage_node statements.

0024-028 *function_name*: **Error creating file** *file_name* **in the system data repository.**

Explanation: Error creating the specified file in the system data repository.

User Response: Check that the system data repository is functioning properly.

0024-029 *function_name*: **This program can not be executed while the filesystem server code is still running.**

Explanation: A program or shell script was invoked which can not be executed while the PIOFS server is running.

User Response: Stop the PIOFS server.

0024-030 *function_name*: **Configuration file** *config_file_name* **not found. Make sure a full pathname is specified.**

Explanation: The configuration file was not found. Make sure a full path name is specified.

User Response: Reissue the command with a valid configuration file.

0024-031 *function_name*: **The network_interface_name** *network_interface_name* **does not correspond to** *nnnnnn*.

Explanation: The network interface name in the configuration file does not correspond to the IPaddresses in the configuration file.

User Response: Reissue the command with a valid configuration file.

0024-032 *function_name*: **There is a limit of** *nnn* **storage nodes.**

Explanation: There is a limit on the number of storage nodes that can be defined.

User Response: Edit the configuration file to make sure a valid number of nodes is specified. Reissue the command with a valid configuration file.

0024-033 *nnnnnn*: **Invalid IPports configuration data.**

Explanation: An invalid port number has been detected. Numbers between 0 and 1024 are reserved for well-known ports.

User Response: Correct the IPports data in the configuration file.

0024-034 *function_name*: **Device statements must have a logical volume size no greater than** *nnn* **Megabytes**

Explanation: The logical volumes built to contain PIOFS data cannot be larger than the specified size.

User Response: Correct the statement.

0024-035 *function_name*: **Cannot copy** *file_name* **to** *directory*.

Explanation: The file *file_name* cannot be copied to *directory*

User Response: Verify that root has write access to *directory* and that there is enough space in *directory* to copy *file_name*.

0024-036 *function_name*: **Cannot extract** *file_name* **from** *shared_library*.

Explanation: Cannot extract object file from shared library.

User Response: Verify permissions on target directory. Verify that there is sufficient free space in target directory.

0024-037 *function_name*: **Cannot modify** *file_name*.

Explanation: Cannot modify target file

User Response: Verify that the target file exists and that it can be written to. Also, verify that the target file is shr.o.

0024-038 *function_name*: **Cannot update** *object*.

Explanation: Cannot replace object in shared library.

User Response: Verify permissions on target directory and shared library.

0024-039 *function_name*: **Effective id must be root to execute** *command*.

Explanation: You attempted to execute the *command* command without root authority. You must be root to execute this command.

User Response: Set effective uid to root.

0024-040 *function_name*: **Cannot fwrite**

Explanation: The **fwrite** could not be performed.

User Response: Verify file permissions.

0024-041 *function_name*: **Cannot open** *file_name* **for read update.**

Explanation: Either the *file_name* file does not exist or permissions do not permit read update.

User Response: Verify that the file exists and that permissions are set to allow reading and updating.

0024-042 *function_name*: *file_name* **not an RS/6000 XCOFF file.**

Explanation: The *file_name* file is not an RS/6000 XCOFF format file.

User Response: Verify the file type.

0024-043 *function_name*: *nnnnnn* **exceeds string table length** *nnnnnn*.

Explanation: The modified string table length exceeds existing string table length.

User Response: None.

0024-044 *function_name*: **Cannot malloc** *nnn* **bytes.**

Explanation: There was insufficient memory to satisfy **malloc** request.

User Response: Make more free memory available, then re-execute the command.

0024-045 *function_name*: **Cannot fseek.**

Explanation: The **fseek.** function could not be performed.

User Response: Verify the file length.

0024-046 *function_name*: **Cannot fread.**

Explanation: The **fread** function could not be performed.

User Response: Verify file permissions.

0024-046 *function_name*: **Symbol** *symbol* **not found.**

Explanation: The symbol, *symbol* was not found in the XCOFF file. This message is also displayed if object file was already modified.

User Response: Verify the version of AIX and the object file.

0024-048 *function_name*: **Storage_nodes must be unique. Storage_node statements have already been specified for hostname =** *hostname*.

Explanation: The storage node statements must be for unique physical nodes.

User Response: Correct the storage node statements.

0024-049 *function_name*: *volume_group* **is not a recognized volume group**

Explanation: The specified volume group is unknown to the system.

User Response: Create the specified volume group or specify a valid volume group.

0024-050 *function_name*: **Volume group (** *volume_group* **) must be varied online**

Explanation: The specified volume group must be varied online.

User Response: Vary the specified volume group online.

0024-200 *function_name*: **User must have root authority to mount the filesystem.**

Explanation: The AIX **mount** command must be issued by the root user.

User Response: Reissue the command as root.

0024-201 *function_name*: **User must have root authority to unmount the filesystem.**

Explanation: The AIX **unmount** command must be issued by the root user.

User Response: Reissue the command as root.

0024-202 *function_name*: **Unable to install the filesystem as a kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to load the kernel extension.

User Response: Check that the **installp** process was successful.

0024-203 *function_name*: **Unable to configure the filesystem kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to configure the kernel extension.

User Response: Check that the **installp** process was successful.

0024-204 *function_name*: **The piofs client daemon failed to start.**

Explanation: The file system requires a client daemon to be active.

User Response: Check that the **installp** process was successful.

0024-205 *function_name*: **A configuration file must be specified as one of the mount options.**

Explanation: The file system requires a configuration file to mount. This configuration file is built when the file system servers are configured with **piofs_config**.

User Response: Ensure that the active configuration files have been copied to the client nodes, this will normally occur automatically if the System Data Repository is running. The files are /**etc**/*config_file_name*:**.active** and /**etc**/*config_file_name***.IPaddresses.active**.

Make sure that the **mount** OPTIONS argument specified the basename of the active configuration file.

0024-206 *function_name*: **Failed reading** /**etc**/*config_file_name*

Explanation: The file system requires a configuration file to mount. This configuration file is built when the file system servers are configured with **piofs_config**.

User Response: Ensure that the active configuration files have been copied to The client nodes, this will normally occur automatically if the System Data Repository is running. The files are /**etc**/*config_file_name***.active** and /**etc**/*config_file_name***.IPaddresses.active**.

Make sure that the **mount** OPTIONS argument specified the basename of the active configuration file.

0024-207 *function_name*: **Failed reading** /**etc**/*config_file_name*

Explanation: The file system requires a configuration file to mount. This configuration file is built when the file system servers are configured via the **piofs_config** shell script.

User Response: Ensure that the active configuration files have been copied to the client nodes, this will normally occur automatically if the System Data Repository is running. The files are /**etc**/*config_file_name***.active** and /**etc**/*config_file_name***.IPaddresses.active**.

Make sure that the **mount** OPTIONS argument specified the basename of the active configuration file.

0024-208 *function_name*: **Mount point** *mount_point* **does not exist.**

Explanation: The directory specified as the mount point does not exist.

User Response: Create the specified directory.

0024-209 *function_name*: **Filesystem already mounted on stub.**

Explanation: The file system is already mounted on the specified mount point.

User Response: Choose a different mount point if you really want to mount the file system more than once.

0024-210 *function_name*: **Unable to unmount, errno =** *nnn*

Explanation: An error was returned from the **uvmount()** system call.

User Response: See the errors returned by the **uvmount()** system call.

0024-211 *function_name*: **Unable to mount, errno =** *nnn*

Explanation: An error was returned from the **vmount()** system call.

User Response: See the errors returned by the **vmount()** system call.

0024-212 *function_name*: **Unable to mount, could not communicate with the servers..**

Explanation: The client code could not communicate with the server code.

User Response: Make sure the server is active.

0024-213 *function_name*: **User must have root authority to issue the** *command* **command.**

Explanation: The command must be issued by the root user.

User Response: Reissue the command as root.

0024-214 *function_name*: **Unable to unconfigure the filesystem kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to unconfigure the kernel extension.

User Response: See the **errno** returned from the **sysconfig()** function.

0024-215 *function_name*: **Unable to uninstall the filesystem, there are still mounted filesystems.**

Explanation: The file system can not be removed from the AIX kernel while there are still mounted Parallel I/O File System file systems.

User Response: Unmount all the Parallel I/O File System file systems.

0024-216 *function_name*: **Unable to uninstall the filesystem, failed to kill piofsclientd.**

Explanation: The Parallel I/O File System client daemon must be killed, before the file system can be removed from the AIX kernel.

User Response: Try killing piofsclientd manually.

0024-217 *function_name*: **Unable to uninstall the filesystem as a kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to remove the kernel extension from storage.

User Response: See the errno returned from the **sysconfig()** function.

0024-218 *function_name*: **Unable to unconfigure the filesystem kernel extension. The filesystem is busy, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to unconfigure the kernel extension. This occurs if the file system has been unmounted, but someone is still in a Parallel I/O File System directory.

User Response: Try to identify the user who is in the Parallel I/O File System directory and have the user change to a different directory.

0024-219 *function_name*: **Unable to unmount. The filesystem is busy, errno =** *nnn*

Explanation: This will occur if somone is still in a Parallel I/O File System directory.

User Response: Try to identify the user who is in the Parallel I/O File System directory and have the user change to a different directory.

0024-220 *function_name*: **Unable to get the vfs number from /etc/vfs.**

Explanation: No entry was found in **/etc/vfs** for the parallel I/O file system.

User Response: Create an entry in **/etc/vfs** for the Parallel I/O File System. The vfs number must be between 8 and 15.

0024-221 *function_name*: **Invalid vfs number in /etc/vfs. The vfs number must be between 8 and 15.**

Explanation: An invalid vfs number was specified in **/etc/vfs**.

User Response: Correct the vfs number in **/etc/vfs**.

0024-222 *nnnnnn*: **The IP address for the specified network_interface_name was not found.**

Explanation: All the servers and clients must communicate on the same network interface. If no network_interface_name is specified in the configuration file, css0 is used.

User Response: Verify the entry for network_interface_name in the configuration file.

0024-223 *function_name*: **Unable to query the** *kernel_extension* **kernel extension, errno=***nnn***.**

Explanation: The AIX **sysconfig()** function failed to query the kernel extension.

User Response: Check that the **installp** process was successful.

0024-224 *function_name*: **This program must be run as ROOT.**

Explanation: Command *command* must be issued by the root user.

User Response: Reissue the command as root.

0024-225 *function_name*: **User** *user_name* **is an invalid user name.**

Explanation: User *user_name* is not known to the system.

User Response: Reissue the command with the correct user name.

0024-226 *function_name*: **User** *user_name* **has already been authorized to use PIOFS.**

Explanation: The userid associated with this user name has already been authorized to use PIOFS.

User Response: None.

0024-227 *function_name*: **I/O error.**

Explanation: The *command* command encountered an I/O error.

User Response: None.

0024-228 *function_name*: **Invalid argument.**

Explanation: An invalid argument was specified on the *command* command.

User Response: Verify the syntax and reissue the command.

0024-229 *function_name*: **Warning: failed to retrieve the configuration file from the system data repository. We will attempt to use the existing configuration file,** *config_file_name***.**

Explanation: This command requires the use of a configuration file. The configuration file could not be retrieved from the system data repository, so the system will attempt to use existing configuration file, *config_file_name*.

User Response: Verify that configuration file, *config_file_name*, is current. If *config_file_name* is not current, undo the command, replace the existing configuration file with the current one, and reissue the command.

0024-230 *function_name*: *user_name* **has a userid = 0. Authorization is provided by default, but filesystem resources will not be tracked.**

Explanation: This user name cannot be explicitly authorized to use PIOFS. This user has been authorized to the file system. However, this user's file system resource usage will not be tracked as is done for regular users.

User Response: None.

0024-400 *function_name*: **Communications sub-system initialization failed. program_name =** *program_name*

Explanation: An error was detected in the communications sub-system initialization routine.

User Response: Collect as much information as is possible about what was happening when this error occurred follow your local procedures for reporting problems.

0024-401 *function_name*: **Could not open file** *file_name*

Explanation: Could not open *file_name*.

User Response: Check that *file_name* has the proper permissions.

0024-402 *function_name*: **Could not memory map file** *file_name*.

Explanation: Could not memory map the MetaData file.

User Response: Collect as much information as possible about what was happening when this error occurred and contact IBM.

0024-403 *function_name*: **Could not read file** *file_name*.

Explanation: Could not read disk data.

User Response: Collect as much information as possible about what was happening when this error occurred and contact IBM.

0024-404 *function_name*: **Could not remove file** *file_name*.

Explanation: Could not remove *file_name*.

User Response: Check that *file_name* has the proper permissions.

0024-405 *function_name*: **Could not rename file** *file_name*.

Explanation: Could not rename *file_name*.

User Response: Check that *file_name* has the proper permissions.

0024-406 *function_name*: **Could not truncate file** *file_name*.

Explanation: Could not truncate *file_name*.

User Response: Check that *file_name* has the proper permissions.

0024-407 *function_name*: **Could not write file** *file_name*.

Explanation: Could not write to file *file_name*.

User Response: Check that *file_name* has the proper permissions.

0024-408 *function_name*: **Internal error number** *nnn* **detected.**

Explanation: An internal error has been detected.

User Response: Collect as much information as is possible about what was happening when the error occurred and follow your local procedures for handling problems.

0024-409 *function_name*: **Invalid configuration data, error number** *nnn*.

Explanation: An invalid data item has been in the configuration file. The possible errors are:

1. Invalid server node number
2. Invalid number of disks, exceeds max disks for a server
3. Invalid number of data blocks

User Response: Correct the item in the configuration file.

0024-410 *function_name*: **Invalid number of disk blocks** *nnn*

Explanation: An invalid number of disk blocks has been detected in the configuration file. It exceeds the servers' limit.

User Response: Correct the number of disk blocks in the configuration file.

0024-414 *function_name*: **The server must be formatted under userid** *userid*.

Explanation: This program must be run under the userid *userid*.

User Response: Reissue the shell script while running under the correct user.

0024-415 *function_name*: **Can not find the active configuration file** *config_file_name*.

Explanation: The configuration file specified on the command line could not be found.

User Response: Make sure the servers have been configured and verify the name of the configuration file that was used to configure the servers. (This should NOT be a fully qualified pathname.)

0024-416 *function_name*: **Unable to rsh to node** *node* **without specifying a password.**

Explanation: Root must be able to **rsh** to all the server nodes without specifying a password.

User Response: Check for Kerberos authentication, a local **.rhosts** file, or the **hosts.equiv** file.

0024-417 *function_name*: **Error converting internet address (***nnnnnn***) into a host name.**

Explanation: A **host.list** file is being built; the **host** command must be accessible.

User Response: Check that the AIX **host** command is accessible.

0024-418 *function_name*: **The server must be started under userid** *userid*.

Explanation: This server must be started under the userid *userid*.

User Response: Reissue the shell script while running under the correct user.

0024-419 *function_name*: **The server has already been started.**

Explanation: The server has already been started.

User Response: None.

0024-420 *function_name*: **The server has already been stopped.**

Explanation: The server has already been stopped.

User Response: None.

0024-421 *function_name*: *nnnnnn* **is not a valid argument.**

Explanation: An invalid argument has been passed to the command.

User Response: Check the usage of the command.

0024-422 *function_name*: **Could not communicate with node** *node*.

Explanation: The specified node failed to respond to the **ping** command.

User Response: Check the specified node.

0024-423 *function_name*: **Unable to start the server on node** *node*.

Explanation: Unable to start the server on the specified node.

User Response: Check to see if the specified node is running.

0024-424 *function_name*: **Unable to stop the server on node** *node*.

Explanation: Unable to stop the server on the specified node.

User Response: Check to see if the specified node is running.

0024-425 *function_name*: **The server must be stopped under userid** *userid*.

Explanation: This command must be run under the userid specified.

User Response: Reissue the shell script running under the correct user.

0024-426 *function_name*: **PIOFS License Problem**
PIOFS Server cannot be started on *node* **because:** *explanation*.

Explanation: There is no license available for the server on the specified node.

User Response: Obtain a license and reissue the command.

0024-427 *function_name*: **PIOFS License Problem**
PIOFS Server on *node* **(pid =** *nnn*) **is operating without a valid license. If you are currently running more PIOFS servers than you have licensed please contact IBM for additional licenses.**

Explanation: There is no license available for the server on the specified node.

User Response: Obtain a license.

0024-428 *function_name*: **PIOFS License Problem**
No license is accessable from node *node*.

Explanation: Neither a license server nor a node lock license for PIOFS is accessable from this node.

User Response: Either place a valid PIOFS node lock license in **/usr/lib/netls/conf/nodelock** or place the appropriate **glb_obj.txt** and **glb_site.txt** files in **/etc/ncs**, which point to an iFOR/LS license server with a valid PIOFS license.

0024-429 *function_name*: **Filesystem is not configured.**

Explanation: A request has been made to start a PIOFS file system but it has not been configured yet.

User Response: Issue **piofs_config** to configure the file system and then reissue the **format_piofs_server** command.

0024-430 *function_name*: **Filesystem is not formatted.**

Explanation: A request has been made to start a PIOFS file system but it has not been formatted yet.

User Response: Issue **format_piofs_server** to format the file system and then reissue the **start_piofs_server** command.

0024-600 *function_name*: **Unable to install Pmp as a kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to load the pmp kernel extension.

User Response: Check that the **installp** process was successful.

0024-601 *function_name*: **Unable to configure the pmp kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to configure the kernel extension.

0024-602 *function_name*: **Unable to uninstall the pmp kernel extension. It is still in use by PIOFS.**

Explanation: The pmp kernel extension cannot be removed from the AIX kernel while it is still being used. The pmp kernel extension is probably being used by a Parallel I/O File System.

User Response: Unmount all Parallel I/O File System file systems and uninstall the Parallel I/O File System kernel extension before uninstalling the pmp kernel extension.

0024-603 *nnnnnn*: **Unable to unconfigure the pmp kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to unconfigure the kernel extension.

User Response: See the errno returned from the **sysconfig()** function.

0024-604 *function_name*: **Unable to uninstall the pmp as a kernel extension, errno =** *nnn*

Explanation: The AIX **sysconfig()** function failed to remove the kernel extension from storage.

User Response: See the errno returned from the **sysconfig()** function.

0024-605 *function_name*: **Unable to uninstall the pmp kernel extension, the Parallel I/O File System filesystem server code is still running.**

Explanation: The pmp kernel extension can not be removed from the AIX kernel while the Parallel I/O File System file system server code is still running.

User Response: Stop the Parallel I/O File System file system server.

0024-606 *function_name*: **Unable to install the pmp kernel extension. The userid is not authorized to install the Pmp kernel extension.**

Explanation: The pmp kernel extension must be installed by root.

User Response: Issue the command while running as root.

0024-607 *function_name*: **Unable to unlink local AF_UNIX domain socket** *nnnnnn* **errno =** *nnn*.

Explanation: The pmp kernel extension uses some local AIX sockets which must be unlinked before reinstalling the kernel extension.

User Response: Unlink the named socket and reissue the command.

0024-608 *function_name*: **Unable to make directory** *directory* **errno =** *nnn*.

Explanation: The pmp kernel extension was unable to make a needed directory.

User Response: Make the directory and reissue the command.

0024-609 *function_name*: **Unable to chmod** *file_name* **errno =** *nnn*.

Explanation: The pmp kernel extension was unable to change the mode of the file *file_name*.

User Response: Correct the error and reissue the command.

0024-610 *function_name*: **Unable to register port number on** *nnnnnn*, **errno =** *nnn*.

Explanation: The port number could not be registered in the services database. The possible errors are:

1. Number provided is a well-known port number
2. Message port number in use
3. Control port number in use
4. piofs-msg is registered under another port number
5. piofs-ctl is registered under another port number

User Response: Correct the error and reissue the command.

0024-800 Unable to open log file *log_file_name* **[***system error message* **]**

Explanation: The error indicated by *system error message* occurred while trying to open *log_file_name*.

User Response: Correct the error and reissue the command.

0024-801 I/O error writing to log file *log_file_name* **[***system error message***]**

Explanation: The error indicated by *system error message* occurred while trying to write to *log_file_name*.

User Response: Correct the error and reissue the command.

0024-802 *node_name* **is responding**

Explanation: A server node that was previously reported as not responding is now responding.

User Response: No action is required.

0024-803 *node_name* **is not responding**

Explanation: The *node_name* server node is not responding.

User Response: Verify that the node is functioning normally.

0024-804 *node_name* **has unexpectedly closed its connection**

Explanation: **piofs_fsck** has unexpectedly lost its connection to server node, *node_name*.

User Response: Verify that the node is functioning normally.

0024-805 *node_name* **wrote unrecognized record**

Explanation: This is an internal error.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-806 *node_name* **gave unexpected parse return value**

Explanation: This is an internal error.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-807 Could not fork *command* **[***system error message***]**

Explanation: The error indicated by *system error message* occurred while trying to issue the *command* command.

User Response: Correct the error and reissue the command.

0024-808 Could not join from *node_name* **[***system error message***]**

Explanation: The error indicated by *system error message* occurred while trying to perform **piofs_fsck** functions.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-809 Select failed [*system error message***]**

Explanation: The error indicated by *system error message* occurred while trying to perform **piofs_fsck** functions.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-810 I/O error when reading from *node_name* **[***system error message***]**

Explanation: The error indicated by *system error message* occurred while trying to read a message from server node *node_name*.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-811 *node_name* **reported wrong node ID**

Explanation: The node number for server node *node_name*, that is specified in the configuration file, is inconsistent with the node number maintained by the server node internally.

User Response: Verify that the node number in the configuration file is correct.

0024-812 *node_name* **reported wrong number of nodes in configuration**

Explanation: The number of nodes, specified in the configuration file, is inconsistent with the value for the number of nodes maintained by the server node internally.

User Response: Verify that the number of nodes specified in the configuration file is correct.

0024-813 Multiple objects have the same name *name*

Explanation: Multiple objects in the file system have the same name. All but one of the objects will be deleted.

User Response: No action is required.

0024-814 Multiple objects have the same ID *ID*

Explanation: Multiple objects in the file system have the same ID. All but one of the objects will be deleted.

User Response: No action is required.

0024-815 Missing object with name *directory_name*

Explanation: There are file system objects with path names under the specified directory name, but the directory itself is not in the file system. The missing directory will be created.

User Response: No action is required.

0024-816 *directory_name* **contains non-existent object** *object_name*

Explanation: Directory, *directory_name*, contain the object, *object_name* but the object is not in the file system. The object will be removed from the directory.

User Response: No action is required.

0024-817 *directory_name* **incorrectly contains object with name** *object_name*

Explanation: Directory, *directory_name*, contain the object, *object_name*, but based on the name of the object, the object does not belong in the directory. The object will be removed from the directory.

User Response: No action is required.

0024-818 *object_name* **is in multiple directories** *object_name*

Explanation: Object, *object_name*, is in multiple directories. It will be removed from all but the appropriate directory.

User Response: No action is required.

0024-819 *file_name* **is missing cell** *node_name* **—** *cell_id*

Explanation: The cell that should be located on the specified server node is missing. It will be created but will contain no data.

User Response: No action is required.

0024-820 *file_name* **has extra cell** *node_name* **—** *cell_id*

Explanation: The cell that exists on the specified server node is extraneous. It will be deleted.

User Response: No action is required.

0024-821 *object_name* **is unreferenced**

Explanation: Object, *object_name*, is not contained in any directory. The object will be inserted into the appropriate directory.

User Response: No action is required.

0024-822 Missing file with ID *object_id*

Explanation: There are cells in the file system for the specified ID, but there is no associated file object. The cells will be deleted.

User Response: No action is required.

0024-823 Correction applied

Explanation: An attempted repair was successful.

User Response: No action is required.

0024-824 Correction failed

Explanation: An attempted repair was unsuccessful.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-825 *object_name* **is not a directory — cannot make correction**

Explanation: An unreferenced object could not be added to a directory because the directory does not exist (probably because the corrective action to create the directory failed).

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-826 Process running on server node *node_name* **unexpectedly terminated**

Explanation: The **piofs_fsck** functions running on server node *node_name* terminated prematurely.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-827 *node_name* **did not understand correction command** *command_name*

Explanation: This is an internal error.

User Response: Reissue **piofs_fsck**. If the problem reoccurs, contact IBM.

0024-828 *object_name* **is not a directory — cannot make correction**

Explanation: An unreferenced object was found, whose parent should be *object_name* However, *object_name* is not a directory. The unreferenced object will be deleted.

User Response: No action is required.

0024-829 Unable to parse config file *config_file_name*

Explanation: The format of the specified configuration file is incorrect.

User Response: Verify the format of the configuration file and reissue the **piofs_fsck** command.

0024-830 Usage: *executable_name* **[–n]** *config_file_name*

Explanation: **piofs_fsck** was invoked incorrectly.

User Response: Verify the arguments.

0024-831 Unable to open *config_file_name*

Explanation: The specified configuration file cannot be opened.

User Response: Verify that the configuration file is accessible.

0024-832 Specified configuration file: *config_file_name* **is not a regular file**

Explanation: The specified configuration file is not a file.

User Response: Verify that the configuration file is accessible and that you are using its correct name.

0024-833 Could not stat() configuration file: *config_file_name*

Explanation: The **stat()** system call for the specified configuration file failed.

User Response: Verify that the configuration file is accessible.

Index

Special Characters

/etc/filesystems 23
  chfs 23
  crfs 23
  editing for the Parallel I/O File System 23
  rmfs 23
/etc/vfs 23
  crvfs 23
  editing for the Parallel I/O File System 23
/usr/lpp/piofs 9

A

access subroutine 126
accessx subroutine 126
adapter type limitation 38
add_piofs_user 40
adding a node or disk 33
adding users 24, 30
administering 29
anonymous FTP server 15
atomicity of reads and writes 85
attributes
  file default 91
authorization to PIOFS
  adding 24, 30
  deleting 30
authorization, deleting 30
authorizing user.
  See add_piofs_user
authorizing users 24, 30

B

base node
  default 91
  for a file 108
BASE_NODE in profile file 91
binding
  libraries 25
block limit 18
  changing 30, 41
  in configuration file 18
  listing 31
block limits, changing
  See change_piofs_user
booting the Parallel I/O File System 30
bsu limit 38
BSUs in profile file 91

C

C examples
  piofs_change_view 103
  piofs_chkpoint 106
  piofs_create 108
  piofs_delete_chkpoint 111
  piofs_fstat 114
  piofs_restore_chkpoint 119
  piofs_statfs 121
C language
  compiling 100
  header files 100
  I/O performance 88
  linking 100
  PIOFS functions 95
  using file offset 92
CAUTIOUS flag
  NORMAL or CAUTIOUS flag 86
  piofs_change_view 86
CAUTIOUS in profile file 91
cell 65
CELLS in profile file 91
cells per server node limit 38
change_piofs_user 41
changing configuration 32
changing view of a Parallel I/O File System file. 101
checkpointing files 89, 105
  copying 89
chfs to edit /etc/filesystems 23
chmod subroutine 129
chown subroutine 133
client node failure 35
client_udp_recvspace in configuration file 19
close subroutine 137
coexistence with other products 5
column partitioning 72
column partitioning example 72
commands 44
  add_piofs_user 40
  change_piofs_user 41
  delete_piofs_user 42
  installpiofs 44
  installpmp 45
  list_piofs_users 46
  piofs_config 47
  piofs_fsck 49
  piofs_verify 51
  start_piofs_server 52
  stat_piofs_server 53
  stat_piofs_user 55
  stop_piofs_server 57

© Copyright IBM Corp. 1995

compiling
C program 100
FORTRAN program 100
libraries 25
concurrent file access 85
configuration 13, 21, 22
changing 32
piofs_config command 22
using SMIT 22
configuration command 47
configuration file 17
block_limit 18
client_udp_recvspace 19
contents 17
devnum 20
hostname 20
IPaddr 20
IPports 19
keywords 18
lnode 20
lvsize 21
network_interface_name 19
object_limit 19
pdisk 21
sample file 18
server_udp_recvspace 19
storage_node 20
syntax 18
udp_sendspace 19
volgrp 21
configuration file sample 18
configuration file, creating 17
copying files
from anonymous FTP server 15
create 107
creating a Parallel I/O File System file 107
creating directory 24
creating the configuration file 17
crfs to edit /etc/filesystems 23
crvfs to edit /etc/vfs 23

D

data transfer limit 38
default file attributes 91
delete the checkpointed version of a Parallel I/O File System file 110
delete_chkpoint 110
delete_piofs_user 42
deleting a node or disk 33
deleting user.
See delete_piofs_user
deleting users 30
determining server status 31
deviation
file renaming 38
deviation *(continued)*
file size limit 38
fpathconf command 37
fsync command 37
hard link 37
memory mapping 37
mkfifo command 37
open flags 37
pathconf command 37
record level locking 37
removal of open directory 38
sync command 37
device number in configuration file 20
devnum in configuration file 20
directory
directory, creating 24
disk
adding or deleting 33
disk space requirements for installation 9
distribution tape installation 13
documentation
iFOR/LS 10
Parallel I/O File System 14
downloading book from anonymous FTP server 15
dup subroutine 141
dup2 subroutine 141

E

environment variable
PIOFS_NOLIC_CMD 33
setting PIOFS_NOLIC_CMD 24
error
client node 35
client or server failure 35
configuration 34
error messages 213
Ethernet-attached limitation 38
examples 86
NORMAL flag 86
piofs_change_view 103
piofs_chkpoint 106
piofs_create 108
piofs_delete_chkpoint 111
piofs_fstat 114
piofs_restore_chkpoint 119
piofs_statfs 121
Existing program with PIOFS 94

F

faccessx subroutine 126
failure
client node 35
server node 35
server PIOFS process 35

fclose subroutine 139
fcntl subroutine 141
fdopen subroutine 149
fflush subroutine 139
ffullstat subroutine 197
fgetc subroutine 161
fgetpos subroutine 157
file
  2GB or larger 92
  2GB or larger, access 93
  access 92
  concurrent access 85
  default attributes 91
  offset 92
  piofs.profile 89
  README 10
  renaming, restriction 38
  seeking 92
  smaller than 2GB 92
file attributes 89
file checkpointing 89
  copying 89
file offset
  PIOFS file offset 92
file permissions 108
file size limit 38
file status 112
file system checker
  See piofs_fsck
files 100
  header files 100
fopen subroutine 149
format server 22
  format_piofs_server command 22
  using SMIT 22
format_piofs_server 43
  commands.format_piofs_server 43
FORTRAN
  compiling 100
  header files 100
  I/O performance 88
  linking 100
  PIOFS functions 97
  using file offset 92
FORTRAN examples
  piofs_change_view 103
  piofs_chkpoint 106
  piofs_create 109
  piofs_delete_chkpoint 111
  piofs_fstat 114, 121
  piofs_restore_chkpoint 119
fpathconf command, non-support 37
fputc subroutine 179
fputs subroutine 182
fread subroutine 154
fread() 88
freopen subroutine 149
fsck
  See piofs_fsck
fseek subroutine 157
fsetpos subroutine 157
fstat subroutine 197
fstatx subroutine 197
fsync command, non-support 37
ftell subroutine 157
FTP
  downloading book 15
fullstat subroutine 197
fwrite subroutine 154
fwrite() 88

G

getc subroutine 161
getchar subroutine 161
getw subroutine 161

H

hard link limitation 37
hardware requirements 9
header files 100
help
  from IBM 36
  online (AIX man pages) 15
hostname in configuration file 20

I

I/O performance 88
IBM, getting help from 36
iFOR/LS
  documentation 10
  files 16
  installation 16
  installation tape 10
  license availability 33
installation
  creating the configuration file 17
  distribution tape 13
  iFOR/LS 16
  verification 17
installation requirements 9
installpiofs 44
installpmp 45
introduction 3
IPaddress in configuration file 20
IPports in configuration file 19

L

libraries 25
license availability 33
licenses 10
licensing, iFOR/LS 16
limit
  basic striping units 38
  BSUs 38
  changing user's 30
  connection type 38
  file size 38
  listing user's 31
  maximum cells per server node 38
  maximum data transfer 38
  maximum path length 38
  maximum server nodes 38
  querying user's 30
limit, user's
  See stat_piofs_user
limitation
  fpathconf command 37
  fsync command 37
  hard link 37
  memory mapping 37
  mkfifo command 37
  open flags 37
  pathconf command 37
  PIOFS disk failure potential 37
  record level locking 37
  risk to permanent files 37
  sync command 37
linking
  C program 100
  FORTRAN program 100
  libraries 25
list_piofs_users 46
listing limits 31
listing users 31
listing users.
  See list_piofs_users
llseek
  to initialize PIOFS offset 94
llseek subroutine 92, 164
lnode in configuration file 20
logical storage number in configuration file 20
logical volume size in configuration file 21
logical volume space 17
lseek restriction with PIOFS file offset 92
lseek subroutine 115, 166
lvsize in configuration file 21

M

man pages 15
memory mapping limitation 37
messages 213
mkdir subroutine 168
mkfifo command, non-support 37
mount 23
mounting a Parallel I/O File System 29

N

Net/LS
  See iFOR/LS
network interface name in configuration file 19
network options 21
node
  adding or deleting 33
non-support
  fpathconf command 37
  fsync command 37
  mkfifo command 37
  pathconf command 37
  sync command 37
non-supported functions 37
NORMAL flag 86
  example 86

O

object 19
object limit
  changing 30, 41
  in configuration file 19
  listing 31
object limits, changing
  See change_piofs_user
offset, file 92
online help (AIX man pages) 15
open directory removal 38
open flags limitation 37
open subroutine 171
overview 3

P

Parallel I/O File System
  man pages 15
Parallel I/O File System-affected subroutines
  access 126
  accessx 126
  chmod 129
  chown 133
  chownx 133
  close 137
  dup 141
  dup2 141
  faccessx 126
  fchown 133

Parallel I/O File System-affected subroutines *(continued)*
fchownx 133
fclose 139
fcntl 141
fdopen 149
fflush 139
ffullstat 197
fgetc 161
fgetpos 157
fopen 149
fputc 179
fputs 182
fread 154
freopen 149
fseek 157
fsetpos 157
fstat 197
fstatx 197
ftell 157
fullstat 197
fwrite 154
getc 161
getchar 161
getw 161
llseek 164
lseek 166
mkdir 168
open. 171
putc 179
putchar 179
puts 182
putw 179
read 184
readv 184
readvx 184
readx 184
remove 190
rename 191
rewind 157
rmdir 194
stat 197
statx 197
unlink 202
write 204
writev 204
writevx 204
writex 204
partitioning
column 72
example of column 72
example of row 75
row 74
path length limit 38
pathconf command, non-support 37
pdisk in configuration file 21
performance, I/O
C language 88
FORTRAN 88
fread() 88
fwrite() 88
general tips 88
setvbuf() 88
physical disk in configuration file 21
PIOFS functions
C language 95
FORTRAN 97
PIOFS server information 31
PIOFS server status 31
piofs_change_view, piofsioctl 101
piofs_chkpoint 89, 105
piofs_config
See configuration command
piofs_create piofsioctl 107
piofs_delete_chkpoint piofsioctl 110
piofs_fsck 49
piofs_fstat piofsioctl 112
piofs_lseek
considerations for use 92
to initialize PIOFS offset 94
piofs_lseek piofsioctl 115
PIOFS_NOLIC_CMD environment variable 24, 33
piofs_restore_chkpoint piofsioctl 118
piofs_statfs 120
piofs_verify 51
piofs.profile
base node 91
BSUs 91
CAUTIOUS 91
cells 91
creation 89
man page 15
syntax 90
USE_PIOFS_LSEEK 91
piofsioctl 85
atomicity of reads and writes 85
piofs_change_view 101
piofs_chkpoint 105
piofs_create 107
piofs_delete_chkpoint 110
piofs_fstat 112
piofs_lseek 115
piofs_restore_chkpoint 118
piofs_statfs 120
profile file
See piofs.profile
Program
Existing, using with PIOFS 94
programming considerations 85
putc subroutine 179

putchar subroutine 179
puts subroutine 182
putw subroutine 179

R

read atomicity 85
read subroutine 184
README file 10
readv subroutine 184
readvx subroutine 184
readx subroutine 184
rebooting the Parallel I/O File System 30
reconfiguration 33
record level locking limitation 37
recovery 35
removal of open directory 38
remove subroutine 190
removing a Parallel I/O File System 32
  from the kernel 32
rename subroutine 191
renaming, file 38
requirements
  hardware 9
  software 9
resources
  querying user's 30
resources, user's
  See stat_piofs_user
restore a checkpoint of a Parallel I/O File System file 118
restore_chkpoint 118
restriction
  fpathconf command 37
  fsync command 37
  hard link 37
  memory mapping 37
  mkfifo command 37
  open flags 37
  pathconf command 37
  PIOFS disk failure potential 37
  record level locking 37
  risk to permanent files 37
  sync command 37
retrieve status record from a Parallel I/O File System file 112
rewind subroutine 157
risk to permanent files 37
rmdir subroutine 194
rmfs to edit /etc/filesystems 23
row partitioning example 75

S

sample configuration file 18
sample piofs.profile file 90
server 22
  format 22
  start 22
  starting 22
server information 31
  See also stat_piofs_server
server node failure 35
server nodes, limit 38
server PIOFS process failure 35
server status 31
  See also stat_piofs_server
server status, information retrieval 120
server_udp_recvspace in configuration file 19
set or determine subfile offset value 115
setvbuf() 88
SMIT 22
software requirements 9
space
  logical volume 17
space, logical volume 17
start server 22
  using SMIT 22
start_piofs_server 52
starting a Parallel I/O File System 29
stat subroutine 197
stat_piofs_server 53
stat_piofs_user 55
statfs 120
status of a server
  See stat_piofs_server
status of a user
  See stat_piofs_user
status of user, querying 30
statx subroutine 197
stop_piofs_server 57
stopping a Parallel I/O File System 29
storage_node in configuration file 20
subfile offset value, set or determine 115
switch-attached limitation 38
sync command, non-support 37
syntax 18, 90
  piofs.profile 90

T

trademarks xi
troubleshooting 35

U

UDP receive buffer size 19
UDP send buffer size 19
udp_sendspace in configuration file 19
unconfiguring a Parallel I/O File System 31

unlink subroutine 202
unmounting a Parallel I/O File System 30
USE_PIOFS_LSEEK
  considerations for use 92
  in profile file 91
user authorization 24, 30
user ids 10
user limits, changing 30
  See also change_piofs_user
user limits, listing 31
user limits, querying 30, 42, 46
user resources, querying 30, 42, 46
user status
  See stat_piofs_user
user status, querying 30, 42, 46

V

verification 17
volgrp in configuration file 21
volume group in configuration file 21

W

ways to access large files 93
write atomicity 85
write subroutine 204
writev subroutine 204
writevx subroutine 204
writex subroutine 204