8

OS-9 System Management

System managers have a range of options to consider. OS-9 allows system managers to tailor their system to the needs of users by changing system modules, setting up the system defaults, etc. OS-9 also allows system managers to maximize the performance of their system by using RAM disks, making bootfiles, making a startup file, etc.

This chapter discusses the following topics of importance to system managers:

- Setting the system defaults using the Init module
- Adding customization modules
- Changing system modules
- Making bootfiles
- Using a RAM disk
- Making a startup file
- Shutting down the system
- Installing OS-9 on a hard disk
- Managing processes in a real-time environment
- Using the tmode and xmode utilities
- Using termcap

Setting Up the System Defaults: the Init Module

The Init module is sometimes referred to as the configuration module. It is a non-executable module located in memory in the OS9Boot file or in ROM. The Init module contains system parameters used to configure OS-9 during startup. The parameters set up the initial table sizes and system device names. For example, the amount of memory to allocate for internal tables, the name of the first program to run (usually either SysGo or shell), an initial directory, etc. are specified. You can examine the system limits in the Init module at any time.

+ **NOTE:** The Init module MUST be present in the system in order for OS-9 to work.

The values in the Init module's table are the system defaults. You can change these defaults in two ways. The first method involves editing the CONFIG macro in the systype.d file. The systype.d file is located in the DEFS directory. After systype.d is edited, the Init module is remade and placed in the new bootfile. The second method involves modifying the Init module with the moded utility. Both methods are discussed later in this chapter. Regardless of the method you use, the changes become the system defaults.

The following is a list of the system defaults listed in the Init module. The term ***offset*** refers to the location of a module field, relative to the starting address of the module. Module offsets are resolved in assembly code by using the names shown here and linking the module with the relocatable library: sys.l or usr.l .

Offset	Name	Description
$30	Reserved	This field is currently reserved for future use.
$34	M$PollSz	***Number of Entries in the IRQ Polling Table*** This is the number of entries in the IRQ polling table. One entry is required for each interrupt generating device control register. The IRQ polling table has 32 entries by default. Each entry in the IRQ polling table is 18 bytes long.
$36	M$DevCnt	***Device Table Size*** This is the number of entries in the system device table. One entry is required for each device in the system. The system device table has 32 entries by default. Each entry in this table is 18 bytes long.
$38	M$Procs	***Initial Process Table Size*** This indicates the initial number of active processes allowed in the system. If this table becomes full, it automatically expands as needed. By default, 64 active processes are allowed. Each entry in the initial process table requires 4 bytes.

Offset	Name	Description
$3A	M$Paths	***Initial Path Table Size*** This is the initial number of open paths in the system. If this table becomes full, it automatically expands as needed. By default, 64 open paths are allowed. Each entry in the initial path table requires 4 bytes.
$3C	M$SParam	***Offset to Parameter String for Startup Module*** This is the offset to the parameter string (if any) to be passed to the first executable module. An offset of zero indicates that no parameter string is required. The parameter string itself is located elsewhere, usually near the end of the Init module.
$3E	M$SysGo	***First Executable Module Name Offset*** This is the offset to the name string of the first executable module; usually SysGo or shell.
$40	M$SysDev	***Default Directory Name Offset*** This is the offset to the initial default directory name string; usually /d0 or /h0. The kernel does a chd and chx to this device prior to forking the initial device. If the system does not use disks, this offset must be zero**.**
$42	M$Consol	***Initial I/O Pathlist Name Offset*** This is the offset to the initial I/O pathlist string. This offset usually points to the /TERM string. This pathlist is opened as the standard I/O path for the initial process. It is generally used to set up the initial I/O paths to and from a terminal. This offset should contain zero if no console device is in use.
$44	M$Extens	***Customization Module Name Offset*** This is the offset to a name string of a list of customization modules (if any). A customization module is intended to complement or change OS-9's existing standard system calls. These modules are searched for during startup. Typically these modules are found in the bootfile. They are executed in system state if found. Modules listed in the name string are separated by spaces. The default name string to be searched for is OS9P2. If there are no customization modules, set this value to zero. **NOTE:** A customization module may only alter the d0, d1, and ccr registers. **NOTE:** Refer to the following section for more information on customization modules.

$46	M$Clock	***Clock Module Name Offset*** This is the offset to the clock module name string. If there is no clock module name string, set this value to zero.
$48	M$Slice	***Timeslice*** The number of clock ticks per timeslice. The number of clock ticks per timeslice defaults to two.
$4A	Reserved	This field is currently reserved for future use.
$4C	M$Site	This is the offset to the installation site code. This value is usually set to zero. OS-9 does not currently use this field.
$50	M$Instal	***Offset to Installation Name*** This is the offset to the installation name string.
$52	M$CPUTyp	***CPU Type*** CPU type: 68000, 68008, 68010, 68020, 68030, 68040, 68070, or 683XX. The default is 68000.
$56	M$OS9Lvl	***Level, Version, and Edition*** This four byte field is divided into three parts: level: 1 byte version: 2 bytes edition: 1 byte For example, level 1, version 2.4, edition 1 would be 1241.

Offset	Name	Description
$5A	M$OS9Rev	***Revision Offset*** This is the offset to the OS-9 level/revision string.
$5C	M$SysPri	***Priority*** This is the system priority at which the first module (usually SysGo or shell) is executed. This is generally the base priority at which all processes start. The default is 128.
$5E	M$MinPty	***Minimum Priority*** This is the initial system minimum executable priority. The default is zero. M$MinPty is discussed later in this chapter and in the ***OS-9 Technical Manual***.
$60	M$MaxAge	***Maximum Age*** This is the initial system maximum natural age. The default is zero. M$MaxAge is discussed later in this chapter and in the ***OS-9 Technical Manual***.

$62	Reserved	This field is currently reserved for future use.
$66	M$Events	***Number of Entries in the Events Table*** This is the initial number of entries allowed in the events table. If the table becomes full, it automatically expands as needed. The default is zero. Each entry in the events table requires 32 bytes. See the ***OS-9 Technical Manual*** for a discussion of event usage. This value is no longer used.
$68	M$Compat	***Revision Compatibility*** This byte is used for revision compatibility. The default is 0. The following bits are currently defined:

Bit 0:	Set to save all registers for IRQ routines.
Bit 1:	Set to prevent the kernel from using stop instructions.
Bit 2:	Set to ignore “sticky” bit in module headers.
Bit 3:	Set to disable cache burst operation (68030 systems).
Bit 4:	Set to patternize memory when allocated or deallocated.
Bit 5:	Set to prevent kernel cold-start from starting system clock.

Offset	Name	Description
$69	M$Compat2	***Compatibility Bit #2*** This byte is used for revision compatibility. The following bits are currently defined:

Bit	Function	
0	0	External instruction cache is *not* snoopy*
	1	External instruction cache is snoopy or absent
1	0	External data cache is *not* snoopy
	1	External data cache is snoopy or absent
2	0	On-chip instruction cache is *not* snoopy
	1	On-chip instruction cache is snoopy or absent
3	0	On-chip data cache is *not* snoopy
	1	On-chip data cache is snoopy or absent
7	0	Kernel disables data caches when in I/O
	1	Kernel *does not* disable data caches when in I/O

* snoopy = cache that maintains its integrity without software intervention.

$6A	M$MemList	***Colored Memory List*** This is an offset to the memory segment list. The colored memory list contains an entry for each type of memory in the system. The list is terminated by a long word of zero. If this field contains a 0, colored memory is not used in this system. For a complete discussion on colored memory, see the ***OS-9 Technical Manual***.
$6C	M$IRQStk	This field contains the size (in longwords) of the kernel's IRQ stack. The value must be 0 or between 256 and $ffff. If the value is zero, the kernel uses a small default IRQ stack. A larger IRQ stack is recommended. The default value is 256 longwords.
$6E	M$ColdTrys	This is the retry counter if the kernel's initial chd to the system device fails. The default value is zero.

+ Throughout this chapter, the system directories referred to are the defaults found in the Init module, unless otherwise specified.

The following is a portion of the distributed init.a file:

```
_INITMOD equ 1 flag reading init module

CPUTyp  set 68000   cpu type (68008/68000/68010)
Level   set 1       OS-9 Level One
Vers    set 2       Version 2.4
Revis   set 3
Edit    set 1       Edition
IP_ID   set 0       interprocessor identification code
Site    set 0       installation site code
MDirSz  set 128     initial module directory size (unused)
PollSz  set 32      IRQ polling table size (fixed)
DevCnt  set 32      device table size (fixed)
Procs   set 64      initial process table size (divisible by 64)
Paths   set 64      initial path table size (divisible by 64)
Slice   set 2       ticks per time slice
SysPri  set 128     initial system priority
```

For more information on the Init module, see the ***OS-9 Technical Manual***.

Customization Modules

Customization modules can be attached to OS-9 during the system's cold-start procedure to increase OS-9's functionality and to allow hardware customization such as special bus arbitration modes. While customization modules extend its capabilities, OS-9 itself is not changed.

NOTE: A customization module may only alter the d0, d1, and ccr registers.

In the Init module, the M$Extens offset points to a list of module names. By default, the name of the list is OS9P2. If the modules are found during cold-start, they are called. If an error is returned, the system stops. Two of these modules are listed here:

- ***Syscache:*** The syscache module allows the system to enable and control any hardware caches present. The default syscache module supplied by Microware controls the on-chip cache(s) for the 68020 and 68030. You can customize this module to take advantage of any external (off-chip) cache hardware the system may have. The syscache module installs the F$CCtl system call routines. If the syscache module is not installed, no system caching takes place.
- ***SSM:*** The system security module (SSM) allows memory protection. The SSM uses the memory management unit (MMU) hardware to grant and deny users permission to access memory.

Changing System Modules

The provided system modules are configured to satisfy the needs of the majority of users. However, you may wish to alter the existing modules or create new modules. You can make new system modules and alterations to existing system modules by either using the moded utility or changing the defaults in the systype.d file. The system modules most commonly altered are the device descriptors and the Init module.

Using the Moded Utility

Use the moded utility to edit individual fields of certain types of OS-9 modules. You can change the Init module and any OS-9 device descriptor modules with moded.

To use the moded utility, type moded, the name of the desired device descriptor, and any options.

The moded: prompt shows that the editor's command mode has been entered.

When moded is invoked, it attempts to read the moded.fields file. moded.fields contains module field information for each type of module to edit. Without this file, moded cannot function.

The provided moded.fields file comes with module descriptions for standard RBF, SBF, SCF, PIPE, NETWORK, UCM, and GFM module descriptors. It also includes a description for the Init module.

To edit the current module, use the e command. If there is no current module, the editor prompts for the module name to edit. The editor prints the name of a field, its current value, and prompts for a new value.

You can enter the following edit commands:

Command	Description
<expr>	A new value for the field
-	Re-display last field
.	Leave edit mode
?	Print edit mode commands
??	Print description of the current field
<cr>	Leave current value unchanged

If the definition of any field is unfamiliar, use the ?? command. This provides a short description of the current field.

Once you have made all necessary changes to the module, exit the edit mode by reaching the end of the module or by typing a period. At this point, the changes made to the module exist only in memory. To write the changes to the actual file, use the w command. This also updates the module header parity and CRC.

NOTE: moded is mainly used for editing existing descriptors. It is somewhat restrictive, and as a result, if you are building a device descriptor or changing a field such as the file manager names, you may not want to use moded.

Complete documentation is available for the moded utility in the ***OS-9 Utilities*** section.

Editing the Systype.d File

The second method of changing system modules requires editing the systype.d file located in the DEFS directory. The systype.d file contains macros such as TERM, DiskH0, etc. for each device descriptor and the Init module. These macros contain basic memory map information, exception vector methods (for example, vectors in RAM or ROM), I/O device controller memory addresses, and initialization data, etc. for each device descriptor and the init module.

The systype.d file consists of five main sections that are used when installing OS-9:

- Init module CONFIG macro
- SCF Device Descriptor macros and definitions
- RBF Device Descriptor macros and definitions
- ROM configuration values
- Target system specific definitions

The CONFIG macro is used when creating the Init module to determine six or more system dependent variables:

Name	Description
MainFram	MainFram is a character string programs such as login use to print a banner which identifies the system. You can modify this string.
SysStart	SysStart is a character string the OS-9 kernel uses to locate the initial process for the system. This process is usually stored in a module called SysGo. Two general versions of SysGo have been provided in the files: SysGo.a for disk-based OS-9 and SysGo_nodisk.a for ROM-based OS-9.
SysParam	SysParam is a character string that is passed to the initial process. This usually consists of a single carriage return.

Name	Description
SysDev	SysDev is a character string containing the name of the path to the initial system disk. The kernel coldstart routine sets the initial execution and data directories to this device prior to forking the SysStart process. Set this label to zero for a ROM-based system. For example, SysDev set 0.
ConsolNm	ConsolNm is a character string that contains the name of the path to the console terminal port. Messages to be printed during startup appear here.
ClockNm	ClockNm is a character string that contains the name of the clock module.

You can set other system parameters in this macro to override the default values created by the init.a source file. This allows you to perform “system tuning” without modifying the generic init.a file.

The following is a portion of an example systype.d file:

```
CONFIG macro

 endm
 Slice set 10
 ifdef _INITMOD
Compat set ZapMem patternize memory
 endc
```

When editing the Init module, constants may use either values or labels. CPUTyp set 68020 is an example of a constant that uses a value. These constants may appear anywhere in the systype.d file. Compat set ZapMem is an example of a constant that uses a label. These constants must appear outside the CONFIG macro and must be conditionalized to be invoked only when init.a is being assembled. If these values are placed inside the CONFIG macro, the old defaults are still used. If a constant that requires a label is placed outside the macro and not conditionalized, illegal external reference errors result when making other files. You can use the _INITMOD label to avoid these errors.

The SCF and RBF device descriptor macro definitions are used when creating device descriptor modules. Five elements are common to SCF and RBF:

Name	Description
Port	Port is the address of the device on the bus. Generally, this is the lowest address that the device has mapped. Port is hardware dependent.
Vector	Vector is the vector that is given to the processor at interrupt time. Vector is hardware/software dependent. Some devices can be programmed to produce different vectors.
IRQLevel	IRQLevel is the interrupt level (1 - 7) for the device. When a device interrupts the processor, the level of the interrupt is used to mask out lower priority devices.
Priority	Priority is the interrupt polling table priority and is software dependent. A non-zero priority determines the position of the device within the vector. Lower values are polled first. A priority of zero indicates that the device desires exclusive use of the vector. OS-9 does not allow a device to claim exclusive use of a vector if another device has already been installed on the vector, nor does it allow the vector to be used by another device once the vector has been claimed for exclusive use.
DriverName	DriverName is the module name of the device driver. This name is determined by the programmer and is used by the I/O system to attach the device descriptor to the driver.

RBF macros may also contain an optional sixth element to describe various standard floppy disk formats. These values are defined in the file rbfdesc.a in the IO directory.

SCF macros contain two additional elements: Parity and BaudRate. The driver uses these values to determine the parity, word length, and baud rate of the device. These values are usually standard codes used by device drivers to access device specific index tables. These codes are defined in the ***OS-9 Technical Manual***.

You should place definitions such as control register definitions that are system specific in systype.d. This allows you to maintain all system specific definitions in a single, system specific file.

Examine the systype.d file. If it does not accurately describe your system, use any text editor to edit the appropriate macro(s) in the systype.d file.

After editing the macro, change your data directory to the IO directory. Use the make utility to generate the required descriptors. For example, the make d0 would generate the descriptors d0 and dd.d0. The output files are placed in the CMDS/BOOTOBJS directory. Include these new descriptors in the bootfile.

NOTE: For more information on the make utility, refer to the chapter on making files and the make utility description in the ***OS-9 Utilities*** section.

Making Bootfiles

A ***bootfile*** contains a list of modules to be loaded into memory during the system's bootstrap sequence. The provided bootfiles have been configured to satisfy the majority of users, but you may want to add or remove modules from an existing bootfile.

Bootlist Files

Bootfiles are usually created using a bootlist file and the -z option of the OS9Gen or TapeGen utilities. The bootlist files contain a list of files, one file per line, to use in creating the bootfile. Using a bootlist file is a convenient way to maintain bootfile contents, as the bootlist file can easily be edited.

The bootlist files are usually located in the CMDS/BOOTOBJS directory, along with the individual files used for constructing the bootfile.

Bootfile Requirements

The contents and module order of a bootfile are usually determined by the end-user's system configuration and requirements. However, note the following points when you construct a bootfile:

- The kernel **MUST** be present in the system, either in ROM or in the bootfile. If the kernel is in the bootfile, **IT MUST BE THE FIRST MODULE**.
- The Init module must be present in the system, either in ROM or in the bootfile.

All other modules are dependent upon the system configuration.

Making RBF Bootfiles

To make a bootfile for an RBF device (hard disk or floppy disk), you need to edit the bootlist file to match your requirements and then run the OS9Gen utility:

```
chd /h0/cmds/bootobjs
<edit bootlist file>
OS9Gen <device> -z=<bootlist>
```

The <device> you specify is the disk that you wish to install the bootfile on. If this device is a hard disk, specify the "format-enabled" device name.

For example, to make a floppy-disk bootfile, type:

```
OS9Gen /d0 -z=bootlist.d0
```

To make a hard disk bootfile, type:

```
OS9Gen /h0fmt -z=bootlist.h0
```

NOTE: Some systems may not support boot files that are greater than 64K in length and/or non-contiguous.

Making Tape Bootfiles

To make a bootfile for an SBF device (tape), you need to edit the bootlist file to match your requirements and then run the TapeGen utility:

```
chd /h0/cmds/bootobjs
<edit bootlist file>
TapeGen /mt0 -bz=bootlist.tape
```

Using the RAM Disk

OS-9 provides support for RAM disks. These disks reside solely in Random Access Memory (RAM). The information stored on a RAM disk can be accessed significantly faster than the same information stored on a hard or floppy disk. Any files may be stored and accessed on a RAM disk.

To use a RAM disk, you must have a device descriptor, a RAM disk driver, and the RBF file manager. You may use multiple RAM disks as long as each RAM disk has a different port address. The only real limitation to the number of RAM disks is the size of the memory. However, some practical considerations exist. For example, using one large RAM disk is more efficient than using many small RAM disks.

In many system configurations, a RAM disk is used as the default system device. When the RAM disk is used as the default system device, it is known as device dd, instead of r0. The name of the device descriptor is dd.r0. Using this descriptor allows compilers to use the RAM disk as a “fast access” device for temporary files, etc. The RAM disk is usually initialized at startup with definition and library files, if it is to be used as the default system device. The init.ramdisk procedure file provided in the root directory accomplishes this.

+ RAM disks may be ***volatile*** or ***non-volatile***. A volatile RAM disk disappears when the system is reset or the power is shut off. A non-volatile RAM disk resides in a place such as battery backed up RAM and does not disappear when the system is reset or powered down.

Volatile RAM disks may be allocated memory either from free system memory or from outside free system memory. The number of volatile RAM disks allocated from free system memory is governed by the port address. There can be up to 1024 different disks, with each disk having a unique address from 0 to 1023.

Volatile RAM disks not allocated from the free system memory must not be part of the system memory list, and they must have a port address greater than or equal to 1024. This port address indicates the actual start address of the RAM disk.

A non-volatile RAM disk may not be located in any memory search list known to the system’s general memory lists. That is, the RAM disk must be “outside” the system’s knowledge. If it is located in a memory search list known to the system’s general memory lists, the RAM disk may be wiped out because the memory is assumed to be un-allocated and may later be given to another module. In addition, the format protect bit must be set for non-volatile RAM disks and the port address must be greater than or equal to 1024.

Making a Startup File

Using bootfiles is not the only way of loading modules and devices into memory at the time of startup. A startup procedure is executed each time OS-9 is booted and the standard SysGo is used. On disk-based systems, the startup procedure executes a startup file. The startup file is located in the root directory of the system disk.

> + The startup file is an OS-9 procedure file. It contains OS-9 commands to be executed immediately after booting the system.

While some modules and devices, such as the kernel, should be loaded from the bootlist file, loading most modules and devices from the startup file can be advantageous. For example, it is easier to upgrade a system by adding modules to the startup file, or the files contained in the startup file. To change these files, you simply use a text editor and make the changes. To change the bootlist file, you must also use the os9gen utility.

Remember: A procedure file is made up of executable commands. Each command is executed exactly as if it were entered from the shell command line. Each line that begins with an asterisk (*) is a comment and is not executed.

From the root directory, you can examine the startup file by entering:

```
$ list startup
```

A listing similar to the following is displayed:

```
-t -np
*
* OS-9
* Copyright 1984 by Microware Systems Corporation
*
* The commands in this file are highly system dependent and should
* be modified by the user.
*
* setime                                ; * start system clock
link shell cio                          ; * make "shell" and "cio" stay in memory
load math                               ; * load math module
* iniz r0 h0 d0 t1 p1                   ; * initialize devices
* load -z=sys/loadfile                  ; * make some utilities stay in memory
* load bootobjs/dd.r0                   ; * get default device descriptor
* init.ramdisk>/nil >>/nil &            ; * initialize it if its the ramdisk
* tsmon /t1 &                           ; * start other terminals
list sys/motd
```

The first executable line, -t -np, turns on the ***talk mode*** option of the shell and turns off the OS-9 prompt option for the duration of this procedure. The talk mode option echoes each executed command to the terminal display. This allows you to see what commands are being executed.

The other executable lines in the distributed startup file are followed by a comment explaining the purpose of the command. Some standard commands are provided as comments. If you want the command executed during the startup procedure, use a text editor to remove the asterisk preceding the command.

For example, to execute the setime command when the startup file is executed, remove the asterisk preceding the command.

NOTE: For systems with battery backed clocks, run setime to start time-slicing, but use the -s option. The date and time will be read from the clock.

Initializing Devices **iniz r0 h0 d0 t1 p1**

The iniz r0 h0 d0 t1 p1 commented command initializes the following specific devices:

r0	RAM Disk
h0	Hard Disk
d0	Floppy Disk
t1	Terminal
p1	Serial Printer

Whenever OS-9 opens a path to a device, it first checks to see if the device is ***known*** to OS-9. To be known, a device must be initialized and memory must be allocated for its device driver. If the device is unknown at the time of the request, OS-9 initializes the device, allocates memory, and opens the path. For example, a simple dir /d0 command initiates this sequence of events if d0 has not been previously initialized.

The iniz utility initializes devices. iniz performs an I$Attach system call on each device name passed to it. This initializes and links the device to the system.

To initialize a device after the system has been started, type iniz and the name(s) of the device(s) to attach to the system. iniz goes through the procedure of initializing the device(s) and allocating the memory needed for the device. If the device is already attached, it is not re-initialized, but the link count is incremented.

For example, to increment the link count of modules, t2 and t3, type:

```
$ iniz t2 t3
```

You can read the device names from standard input with the -z option or from a file with the -z=<file> option. To increment the link counts of devices listed in a file called /h0/add.files, type:

```
iniz -z=/h0/add.files
```

You can use the deiniz utility to close a path to a device. deiniz checks the link count before removing the device from storage. If the link count is greater than one, deiniz lowers the link count. If the link count is one, deiniz lowers the link count, making it zero, and removes the device from the system device table. The device then becomes ***unknown*** to OS-9.

To use the deiniz utility, type deiniz followed by the name(s) of the devices(s) to be removed from the system.

For example, to decrement the link count of module p2, type:

```
$ deiniz p2
```

deiniz can read the device names from standard input with the -z option or from a file with the -z=<file> option. To remove the files listed in a file called /h0/not.needed, type:

```
$ deiniz -z=/h0/not.needed
```

+ This initialize/de-initialize sequence can result in slower execution of programs and could cause memory fragmentation problems. To avoid these symptoms, Microware recommends that all devices connected to the system at startup be iniz-ed in the startup file.

NOTE: Non-sharable devices must be placed in a bootfile to become known to the system. If a non-sharable device is iniz-ed, it is unusable because the link count will have been incremented, causing it to appear to be in use.

iniz-ing the connected device at startup initializes the device and allocates memory for its driver for the duration of the time that the system is running, unless specifically deiniz-ed. For example, a system with two floppy drives and one hard disk would iniz these devices in the startup file:

```
iniz h0 d0 d1 t1 p1 p
```

NOTE: For more information on the iniz and deiniz utilities, refer to the ***OS-9 Utilities*** section.

Loading Utilities Into Memory load -z=sys/loadfile

The next line of the startup file loads a number of utilities into memory. If a utility is not already in memory, it must be loaded into memory before it is used. Pre-loading basic utilities at startup time avoids the necessity of loading the utility each time it is executed.

To load utilities into memory at startup, you must create a file containing the names of each utility to load, one utility per line. While the file may have any desired name, Microware recommends loadfile for obvious reasons. This file can be located in any directory as long as you specify its location on the command line. If loadfile were located in the SYS directory, the startup file command line is:

load -z=sys/loadfile

Previous versions of the Professional OS-9 package had the following commented line in the startup file:

load utils

This method involved creating a utils file by merging the desired utilities into a single file in the commands directory. While this method may still be used, using loadfile is preferable because it uses less disk space and is easier to edit.

Loading the Default Device Descriptor load bootobjs/dd.r0

Many OS-9 compilers and application programs look for definition files and libraries in directories located on the default system device. The default system device is known as dd. dd may be defined as any disk device, but it is usually synonymous for one of the following devices:

r0	RAM Disk
h0	Hard Disk
d0	Floppy Disk

If a default device is to be used (dd) and the device descriptor is not in the bootfile, then you must load the device descriptor. The next line in the startup file loads the device descriptor. The default device is the RAM disk named r0. If you want another device to be the default device descriptor, change the .r0 extension to reflect the appropriate device. If you have a dd device in your bootfile or if no default device is to be used, leave this line as a comment.

Initializing the RAM Disk **init.ramdisk>/nil >>/nil &**

If you are going to use the RAM disk, a library and definition file structure may be built on the RAM disk. The next line in the startup file executes the init.ramdisk procedure file. init.ramdisk is located in the root directory. It sets up LIB and DEFS directories on /dd. To name the RAM disk /r0, you must change a single line in init.ramdisk; change chd /dd to chd /r0.

NOTE: RAM disks are discussed elsewhere in this chapter.

Multi-User Systems **tsmon /t1 &**

The tsmon utility is used to make your system a multi-user system. This utility supervises idle terminals and initiates the login procedure for multi-user systems. The startup file command line: tsmon /t1& initiates the time-sharing monitor on the serial port /t1.

tsmon can monitor up to 28 device name pathlists. Therefore, if you have multiple devices for tsmon to monitor, you can specify up to 28 devices on each tsmon command line. You can use the ex built-in shell command to execute tsmon without creating another shell. This conserves system memory. For example:

```
ex tsmon term t1 t2 t3 t4 t5&
```

When a carriage return is typed on any of the specified paths, tsmon automatically forks login and standard I/O paths are opened to the device.

The login procedure uses the password file located in the SYS directory for individual login validation. The provided password file has two example login entries. Each of the fields in an entry in the password file is explained in the chapter on the shell and in the login utility description in the ***OS-9 Utilities*** section. If login fails because you could not supply a valid user name or password, control returns to tsmon.

For more information on the tsmon utility, refer to the ***OS-9 Utilities*** section.

System Shutdown Procedure

There will be times when, for one reason or another, you want to bring your system down. When you reset or power down your system, you may need to do more than just press the reset button. Certain programs need to be shut down gracefully. For example, most network communications, print spoolers, and inter-system processes need special attention. These processes may have options or other arrangements that need to be considered before shutting down your system.

In addition to taking care of processes that require special attention, you should prepare the system's users for the shutdown. If at all possible, allow users enough time to save their files and get off the system. One way of alerting users that the system is going down is by echoing a message using the echo and tee utilities. However, you should realize that messages sent over the system in this manner will not be seen by users who do not press a carriage return after the message has been sent. For example, if a programmer is sitting at a shell prompt, the message will not appear on the terminal screen until a carriage return is entered.

> + In this case, verbal warnings are important. This means that in addition to sending a warning message out over the system, you may want to use either an intercom system or the telephone to talk to each person connected to the system.

You can simplify the process of actually shutting down your system by creating a procedure file. Once created, you can run the procedure either from the shell command line prompt or you can create a separate password entry for the sole purpose of shutting down the system.

For example, if you have a procedure file called shutdown.sys, you could create the following password file entry:

```
sys,shutdown,0.0,128,.,sys,shell shutdown.sys
```

Once you login as user sys with password shutdown, the shut down procedure begins because the system immediately has the shell execute the shutdown.sys file.

The following is an example of a useful procedure file for shutting down the system:

```
-t -nx -np
*
* System Shutdown Procedure
*
echo WARNING The system will shut down in 3 minutes ! tee /t1 /t2 /t3 /t4 /t5
sleep -s 60
echo WARNING The system will shut down in 2 minutes ! tee /t1 /t2 /t3 /t4 /t5
sleep -s 115
echo WARNING  5 seconds to system shut down ! tee /t1 /t2 /t3 /t4 /t5
sleep -s 5
spl -$                                  ; * terminate spooler
nmon /n0 -d                             ; * shutdown network
sleep -s 3                              ; * wait 3 seconds
break                                   ; * call ROM debugger
```

The first six commands after the comment identifying the function of the procedure broadcast three warnings to the terminals on the system. The first warning tells the users that the system is going down. The other two warnings serve as reminders. Remember that you should also give verbal warnings.

The remaining command lines shut down the system:

spl -$	This command terminates the spooler. All unfinished jobs are lost when the spooler is terminated.
nmon /n0 -d	This command brings the network down. Users from other networks will no longer be able to login to the system being shut down.
sleep -s 3	This command causes the system to wait three seconds before executing the next command line. This allows the previous commands time to complete execution.
break	This command sends a break call to the ROM debugger. When the ROM debugger receives this call, the system shuts down.

Installing OS-9 On a Hard Disk

Once you have brought up the system and tested its basic operations, install OS-9 on the hard disk and use it as the system boot device. Installing the distribution software on the hard disk involves five steps:

- Checking the hard disk device descriptor
- Formatting the hard disk
- Copying the distribution software on to the hard disk
- Making the hard disk the system boot disk
- Test-booting from the hard disk

Checking the Hard Disk Device Descriptor

The installed hard disk may not necessarily match the parameters in the provided /h0 and /h0fmt device descriptors. For example, the number of cylinders, heads, etc. for your hard disk may be different than the default parameters specified in the device descriptors. Before attempting to use the hard disk, carefully examine the disk macros in systype.d.

If the parameters match the drive in use, the supplied descriptors will work. If not, edit systype.d and remake the descriptors or use the moded utility to remake the descriptors. The moded utility makes/changes any device descriptor module and updates its CRC.

Once the descriptors are made, make a new bootfile with the new descriptors replacing the old ones.

Formatting the Hard Disk

Once the descriptors match the type of drive in use, format the hard disk. Formatting the hard disk builds an OS-9 file structure on the media and tests the media for defective areas. Any new descriptors are also checked.

WARNING: If you have any vital information such as data or programs on this disk, you should perform backups to floppy or tape of this information. The format process completely erases any data on the disk.

To turn off page pause and format the hard disk, enter:

```
$ tmode nopause
$ format /h0fmt -c=<cluster size>
```

NOTE: /h0fmt *must* be the device name, as /h0 is format protected. Use the -c option for large drives only.

The format utility asks whether you want to perform a physical format and a physical verify. Answer y to both questions. The physical format operation is a lengthy process. The larger your hard disk is, the longer you can expect to wait. The logical verify reads each cluster from the disk.

Copying the Distribution Software on to the Hard Disk

Once the hard disk has been formatted correctly, use the dsave utility to copy the distribution software on to the hard disk.

To copy the distribution files:

¿ Insert the first system disk in drive /d0. The first system disk contains the CMDS directory.

¡ Change your current data directory to /d0:

```
$ chd /d0
```

¬ Copy all files from /d0 to /h0:

```
$ dsave -eb50 /h0
```

If you have more than one floppy disk to copy:

Ð Remove the disk in /d0 and replace it with the new disk to copy.

ƒ Change your execution directory to /h0/CMDS:

```
$ chx /h0/cmds
```

The hard disk is now your current execution directory.

Ý Copy all files from /d0 to /h0:

```
$ dsave -eb50 /h0
```

Repeat this step until all floppy disks have been copied to the hard disk.

NOTE: The first disk copied to the hard disk is the distribution disk containing the CMDS directory.

Making the Hard Disk the System Boot Disk

Copying files on to the hard disk installs the software on the hard disk. It *does not* make the hard disk a bootable disk. To make the hard disk the system boot disk, use the os9gen utility.

The OS9Boot file is distributed with your system software. An OS9Boot.h0 bootfile may also be included. The only difference between these files is the default system device name string in the Init module. OS9Boot refers to /d0, while OS9Boot.h0 refers to /h0.

Assuming that these files have been copied on to the hard disk, do the following to make the hard disk bootable:

¿ Change your current data directory to /h0:

```
$ chd /h0
```

¡ Rename OS9Boot to retain a copy to use with a floppy system:

```
$ rename OS9Boot OS9Boot.d0
```

¬ Make the hard disk bootable with the correct bootfile. **NOTE:** You must specify /h0fmt as the device.

```
$ os9gen /h0fmt OS9Boot.h0
```

Test Booting from the Hard Disk

Once you have completed the above steps, test that the system actually boots from the hard disk.

If the system fails to boot correctly, reboot the system. Carefully examine the results of the actions previously described.

Managing Processes in a Real-time Environment

The ability to manage processes in a real-time environment is one of OS-9's advantages. OS-9 has three main methods by which system managers can manage processes in a real-time environment:

- Manipulating process' priority
- Using D_MinPty and D_MaxAge to alter the system's process scheduling
- Having system state processes and user state processes

Manipulating Process' Priority

When you execute processes on the command line, you can change their initial priorities using the process priority modifiers discussed in the chapter on the shell. This allows you to set the priority on crucial tasks higher so that they run sooner and more often than processes that are less crucial.

NOTE: The initial priority is also a parameter for the fork and chain system calls.

Using D_MinPty and D_MaxAge to Alter the System's Process Scheduling

The way OS-9 schedules processes can be affected by the D_MinPty and D_MaxAge system global variables. D_MinPty and D_MaxAge are available to super users through the F$SetSys system call. These system variables can be used to effect the aging of processes. **Remember:** A process' initial priority is aged each time it is passed by for execution while it is waiting for CPU time.

D_MinPty defines a minimum priority below which processes are neither aged nor considered candidates for execution. Processes with priorities less than D_MinPty remain in the waiting queue and continue to hold any system resources that they held before D_MinPty was set.

+ D_MinPty is usually set to zero. All processes are eligible for aging and execution when this value is set to zero because all processes have an initial priority greater than zero.

If you have a critical process that needs to be run and several other users have processes that they want to run, use the process priority modifier to increase the priority of the critical process. Then, set D_MinPty to a value that is less than the priority you assigned to the critical process but greater than the priority of the other processes. The critical process now continues using the CPU until another process with a priority greater than D_MinPty is entered into the waiting queue or the critical process is finished.

For example, if D_MinPty is set to 500 and you set the priority of your process at 600, your process continues to use the CPU while processes with priorities less than 500 cannot run until D_MinPty is reset.

CAVEAT: D_MinPty is potentially dangerous. If the minimum system priority is set above the priority of all running tasks, the system will completely shut down and can only be recovered by a reset. It is crucial to restore D_MinPty to zero when the critical task finishes or to reset D_MinPty or a process' priority in an interrupt service routine.

+ D_MaxAge defines a maximum age over which processes are not allowed to mature. By default, this value is set to zero. When D_MaxAge is set to zero, it has no effect on the processes waiting to use the CPU.

When set, D_MaxAge essentially divides tasks into two classes: ***low priority*** and ***high priority***. A low priority task is any task with a priority below D_MaxAge. Low priority tasks continue aging until they reach the D_MaxAge cutoff, but they are not executed unless there are no high priority tasks waiting to use the CPU.

A high priority task is any task with a priority above D_MaxAge. A high priority task will receive the entire available CPU time, but it will not be aged. When the high priority task(s) are inactive, the low priority tasks are run.

For example, if D_MaxAge is set to 2000 and three processes with initial priorities of 128 are in the active queue, the processes run just as if D_MaxAge had not been set. Then, if a process with an initial priority of 2500 is entered into the active queue, it receives CPU time when the process currently in the CPU has finished. Once using the CPU, the high priority process runs uninterrupted until a process with a higher priority enters the active queue or the process finishes. When the process finishes executing, the low priority processes will again be able to use the CPU.

NOTE: Any process performing a system call is not pre-empted until the call is finished, unless the process voluntarily gives up its timeslice. This exception is made because these processes may be executing critical routines that affect shared system resources and could be blocking other unrelated processes.

Using System-State Processes and User-State Processes

The second method that OS-9 uses to manage real-time priority processing is the existence of system-state processes. System-state processes are processes running in a supervisor or protected mode. System-state processes basically have unlimited access to system memory and other resources. When a process in system state wants to use the CPU, it waits until it has the highest age. Once it is available to use the CPU, a process in system state runs until it finishes instead of running for a specified timeslice.

Processes that are in user state do not have access to all points in memory and do not have access to all of the commands. When a process in user state gains time in the CPU, it runs only for the time specified by the timeslice. When it finishes using its timeslice, it is entered back in the waiting queue according to its initial priority.

Using the Tmode and Xmode Utilities

The tmode and xmode utilities are also available to help you customize OS-9. Use the tmode utility to display or change the operating parameters of the user's terminal. tmode affects open paths, not the device descriptor itself, so the changes made by it are temporary. The changes made by tmode are inherited if the paths are duplicated, but not if the paths are opened explicitly.

The xmode utility is similar to tmode. Use xmode to display or change the initialization parameters of any SCF-type device such as a video display, printer, RS-232 port, etc. xmode actually updates the device descriptor. The change persists as long as the computer is running even if paths to the device are repetitively opened and closed. Some common uses of xmode are to change the baud rates and control definitions.

In SSM systems, you must have write permission for the descriptor module in order for xmode to work. You can use the fixmod utility to change the permissions.

NOTE: tmode and xmode work only on SCF and GFM devices.

Using the Tmode Utility

To use the tmode utility, type tmode and any parameter(s) to change. If you give no parameters, the present values for each parameter are displayed. Otherwise, the parameter(s) given on the command line are processed. You can give any number of parameters on a command line. Use spaces or commas to separate each parameter.

If a parameter is set to zero, OS-9 no longer uses the parameter until it is re-set to a code OS-9 recognizes. For example, the following command sets xon and xoff to zero:

```
tmode xon=0 xoff=0
```

Consequently, OS-9 will not recognize xon and xoff until the values are re-set.

To re-set the values of a parameter to their default as given in this manual, specify the parameter with no value.

Use the -w=<path#> option to specify the path number affected. If a path number is not provided, standard input is affected.

NOTE: If you use tmode in a shell procedure file, you must use the -w=<path#> option to specify one of the standard paths (0, 1, or 2) to change the terminal's operating characteristics. The change remains in effect until the path is closed. To effect a permanent change to a device characteristic, you must change the device descriptor. You may alter the device descriptor to set a device's initial operating parameters using the xmode utility.

Five parameters need driver support in order to be changed by tmode: type, par, cs, stop, and baud. If you try to change these parameters without driver support, tmode has no effect.

The tmode parameters are documented in the ***OS-9 Utilities*** section.

Using the Xmode Utility

To use the xmode utility, type xmode and any parameter(s) to change. If you give no parameters, the present values for each parameter are displayed. Otherwise the parameter(s) given on the command line are processed. You can give any number of parameters on a command line. Use spaces or commas to separate each parameter. You must specify a device name if the given parameter(s) are to be processed.

Like tmode, if a parameter is set to zero, the device no longer uses the parameter until it is re-set to a recognizable code. To re-set the values of parameters to their default, specify the parameter with no value. This re-sets the parameter to the default value as given in this manual.

Five parameters require further explanation: type, par, cs, stop, and baud. xmode changes these parameters only if the device is iniz-ed directly after the xmode changes and the driver supports these changes. Changing these parameters is usually done in the startup file or by first deiniz-ing a file. For example, the following command sequence changes the baud rate of /t1 to 9600:

```
$ deiniz t1
$ xmode /t1 baud=9600
$ iniz t1
```

This type of command sequence changes the device descriptor and initializes it on the system. Only the five parameters mentioned above need this special sequence to be changed. All other xmode parameters are changed immediately.

xmode's parameters are documented in the ***OS-9 Utilities*** section.

The Termcap File Format

The termcap file is a text file that contains control code definitions for one or more types of terminals. Each entry is a complete description list for a particular kind of terminal.

The first section of a termcap entry is divided into three parts.

- A two character entry
- The most common name
- A long name

Each part is a different way of naming the terminal. A **|** character separates the parts of a termcap entry. The first part is a two character entry. This is a holdover from early UNIX editions. The second part is the most common name for the terminal. This name must contain no blanks. The final part is a long name fully describing the terminal. This name may contain blanks for readability. For example:

```
kh|abm85h|kimtron abm85h:
```

The TERM environment variable must be set to the name used in the second part of the name section. In the following example, TERM is set to abm85h:

```
$ setenv TERM abm85h
```

NOTE: You can check the values stored in TERM by using the printenv command:

```
$ printenv
TERM=abm85h
```

The rest of the entry consists of a sequence of control code specifications for each control function. Use a colon (:) character to separate each item in the list. You can continue an entry on to the next line by using a backslash (\) character as the last character of the line. It must appear after the last colon of the previous item. The next line must begin with a colon. For example:

```
ka|amb85|kimtron abm85:\
:ct=\E3: ...
```

Each item begins with a terminal ***capability***. Each capability is a two character abbreviation. Each capability is either a boolean itself or it is followed by a string or a number. If a boolean capability is present in the termcap entry, then the capability exists on that terminal.

All numeric capabilities are followed by a pound sign (#) and a number. For example, the number of columns capability for an 80 column terminal could be described as follows:

```
co#80:
```

All string capabilities are followed by an equal sign (=) and a character string. You can enter a time delay in milliseconds directly after the equal sign (=) if padding is allowed in that capability. The padding characters are supplied by tputs() after the remainder of the string is transmitted to provide the time delay. The time delay may be either an integer or a real. The time delay may be followed by an asterisk (*). The asterisk specifies that the padding is proportional to the number of lines affected.

NOTE: It is often useful to specify the time delay using the real format. For example, the clear screen capability is specified as ^z with a time delay of 3.5 milliseconds by the following entry:

```
cl=3.5*^z:
```

Escape sequences may be indicated by an \E to indicate the escape character. A control character is indicated by a circumflex (^) preceding the character. The following special character constants are supported:

\b	Backspace	($08)
\f	Formfeed	($0C)
\n	Newline	($0A)
\r	Return	($0D)
\t	Tab	($09)
\\	Backslash	($5C)
\^	Circumflex	($5E)

Characters may be specified as three Octal digits after a backslash (\). For example, if a colon must be used in a capability definition, it must be specified by \072. If it is necessary to place a null character in a capability definition use \200. C routines using termcap strip the high bits of the output, therefore \200 is interpreted as \000.

Termcap Capabilities

The following is a list of termcap capabilities recognized by termcap. Not all of these capabilities need to be present for most programs to use termcap. They are provided for completeness. (P) indicates that padding may optionally be specified. (P*) indicates that the optional padding may be based on the number of lines affected:

Name	Type	Padding	Description
ae	string	(P)	End alternate character set
al	string	(P*)	Add new blank line
am	boolean		End alternate character set
as	string	(P)	Start alternate character set
bc	string		Backspace if not ^H
bs	boolean		Terminal can backspace with ^H
bt	string	(P)	Back tab
bw	boolean		Backspace wraps from column 0 to last column
CC	string		Command character in prototype if terminal settable
cd	string	(P*)	Clear to end of display
ce	string	(P)	Clear to end of line
ch	string	(P)	Horizontal cursor motion only, line stays some
cl	string	(P*)	Clear screen
cm	string	(P)	Cursor motion
co	numeric		Number of columns in line
cr	string	(P*)	Carriage return (default ^M)
cs	string	(P)	Change scrolling region (VT100), like cm
cv	string	(P)	Vertical cursor motion only
da	boolean		Display may be retained above
dB	numeric		Number of milliseconds of backspace delay needed
db	boolean		Display may be retained below
dC	numeric		Number of milliseconds of carriage return delay needed
dc	string	(P*)	Delete character
dF	numeric		Number of milliseconds of formfeed delay needed
dl	string	(P*)	Delete line

Name	Type	Padding	Description
dm	string		Delete mode (enter)
dN	numeric		Number of milliseconds of newline delay needed
do	string		Down one line
dT	numeric		Number of milliseconds of tab delay needed
ed	string		End of delete mode
ei	string		End insert mode **NOTE:** If ic is used, enter :ec=:
eo	string		Can erase overstrikes with a blank
ff	string	(P*)	Hardcopy terminal page eject (default ^L)
hc	boolean		Hardcopy terminal
hd	string		Half-line down (1/2 linefeed)
ho	string		Home cursor (if no cm)
hu	string		Half-line up
hz	string		Hazeltime: cannot print tildas (~)
ic	string	(P)	Insert character
if	string		Name of file containing initialization string
im	boolean		Insert mode (enter). **NOTE:** If ic is specified use :im=:
in	boolean		Insert mode distinguishes nulls on display
ip	string	(P*)	Insert pad after character inserted
is	string		Terminal initialization string
k0-k9	string		Sent by other function keys 0-9
kb	string		Sent by backspace key
kd	string		Sent by down arrow key
ke	string		Take terminal out of ***keypad transmit*** mode
kh	string		Sent by home key
kl	string		Sent by left arrow key
kn	numeric		Number of other keys
ko	string		Termcap entries for other non-function keys
kr	string		Sent by right arrow key
ks	string		Put terminal in keypad transmit mode
ku	string		Sent by up arrow key

Name	Type	Padding	Description
l0-l9	string		Labels on other function keys
li	numeric		Number of lines on screen or page
ll	string		Last line, first column (if no cm entry)
ma	string		Arrow key map
mi	boolean		OK to move while in insert mode
ml	string		Memory lock on above cursor
ms	boolean		OK to move while in standout and underline mode
mu	string		Turn off memory lock
nc	boolean		Carriage return down not work
nd	string		Non-destructive space
nl	string	(P*)	Newline character
ns	boolean		Terminal is a non-scrolling CRT
os	boolean		Terminal overstrikes
pc	string		Pad character (rather than null)
pt	boolean		Has hardware tabs
se	string		End stand out mode
sf	string	(P)	Scroll forwards
sg	numeric		Number of blank characters left by se or so
so	string	(P)	Begin stand out mode
sr	string	(P)	Scroll reverse
ta	string		Tab (other than ^I or without padding)
tc	string		Entry of terminal similar to last termcap entry
te	string		String to end programs that use cm
ti	string		String to begin programs that use cm
uc	string		Underscore one character and move past it
ue	string		End underscore mode
ug	numeric		Number of blank characters left by us or ue
ul	boolean		Terminal underlines but doesn’t overstrike
up	string		Upline (cursor up)
us	string		Start underscore mode

Name	Type	Padding	Description
vb	string		Visible bell
ve	string		Sequence to end open/visual mode
vs	string		Sequence to start open/visual mode
xb	boolean		Beehive terminal (f1=<esc>, f2=^C)
xn	boolean		Hewline is ignored after wrap
xr	boolean		Return acts like ce \r\n
xs	boolean		Standout not erased by writing over it
xt	boolean		Tabs are destructive

Of the capabilities, the most complex and important capability is cm: cursor addressing. The string specifying the cursor addressing is formatted similar to the C function: printf(). It uses % notation to identify addressing encodings of the current line or column position. The line and the column being addressed could be considered the arguments to the cm string. All other characters are passed through unchanged. The following is the notation used for cm strings:

%d	a decimal number (origin 0)
%2	same as %2d
%3	same as %3d
%.	ASCII equivalent of value
%+x	adds x to value, then %
%>xy	if value > x adds y, no output
%r	reverses the order of row and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140
%B	BCD (16*(x/10) + (x%10), no output
%D	reverse coding (x-2*(x%16)), no output

The following examples illustrate the use of the preceding notations:

cm=6\E&%r%2c%2Y:	This terminal needs a 6 millisecond delay, rows and columns reversed, and rows and columns to be printed as two digits. The <esc>& and Y are sent unchanged. **(HP2645)**

cm=5\E[%i%d;%dH:	This terminal needs a 5 millisecond delay, rows and columns separated by a semicolon (;), and because of its origin of 1, rows and columns are incremented. The <esc>[, ; and H are transmitted unchanged. **(VT100)**
cm=\E=%+ %+ :	This terminal uses rows and columns offset by a blank character. **(ABM85H)**

Example Termcap Entries

```
ka|abm85|kimtron abm85:\
    :ce=\ET:cm=\E=%+ %+ :cl=^Z:\
    :se=\Ek:so\Ej:up=^K:sg#1
```

If two entries in the same termcap file are very similar, one can be defined as identical to the other with certain exceptions. To do this, tc is used with the name of the similar terminal. This capability must be the last in the entry. All exceptions to the other terminal must appear before the tc listing. If a capability must be cancelled, use <cap>@. For example, this might be a complete entry:

```
kh|abm85h|kimtron abm85h:\
    :se=\EG0:so\EG4:tc=abm85:
```

End of Chapter 8

NOTES