5.2 Version

GUIDE TO THE OPERATING SYSTEM

July 23, 1982

Introduction 1

Configuration 1

OS Volume Types 2

System Files 3

Installing the OS 4

How to Boot the OS 6

Shutting Down the OS 7

Program Development 8

The OS Command Shell 9

The Filer 11

The Privileged Filer 14

The Asynchronous File System 16

OS Performance 16

Printers and RS232 Input/Output 17

Stack Size 20

Intrinsic Units 21

Paslib 22

The Pascal Heap 25

The OS Interface 27

Additions to the OS 43

OS Error Messages 47

INTRODUCTION

This document explains how to use the 5.2 release of the Operating System. It includes all relevant material from prior releases and also describes the new features of this release. The User's Guide does not explain each feature of the OS in detail; instead, it explains operations such as installing and booting the system, and presents the details of the Command Shell and Filer commands (see the Operating System Reference Manual for a complete description of the Operating system).

Because the Operating System Reference Manual is not updated for each new release, the User's Guide also contains explanations of any new features or calls that may not fit into the above topics . The User's Guide assumes that the Operating System Reference Manual that you have is dated March 1, 1982.

This release of the Operating System boots from a Profile or Twiggy rather than the Apple II. The standalone OS is installed and operates differently than the OS that boots from the Apple in several ways. Please read the paragraphs below that explain the environment that this version of the OS supports before attempting to install and run it.

CONFIGURATION

The standalone OS boots from either a Profile or a Twiggy. Most people will boot the standalone OS from a Profile, and use another hard disk for the Monitor.

The standalone OS restricts how you can use the various devices. The OS treats the Corvus as a source of Monitor files, not as an OS volume; the Apple, the Disk-II floppies and the Sanyo screen are totally inaccessible from the OS; the Lisa screen and keyboard are reserved for Applications. Therefore, readlns and writelns can only be seen on your Soroc.

WARNING: YOU NEED A SOROC, TO USE THE STANDALONE OS! Attach the Soroc to channel A of the Lisa; this channel is the second from the left when you are standing in front of the system.

When the OS boots from a Profile, that Profile must be attached to the parallel port (the connector farthest to the right when viewed from the front). This port is 'PARAPORT' to the OS and '&3' to the Monitor.

Your other hard disks are attached to the N-Port card, which must (for now) be in Slot 2 (the middle slot). Starting from the bottom of the N-Port card, the ports are named '&4', '&5', and '&6' by the monitor, and 'SLOT2CHAN1', 'SLOT2CHAN2', and 'SLOT2CHAN3' by the OS. The monitor gives preference to disks attached to the bottom of the N-Port card, so your monitor disk should probably be attached to &4.

Before installing the OS, you must run the OSCONFIG program under the Monitor. OSCONFIG produces a configuration file that defines, at boot time, which devices are attached at each port (&3 thru &6), and which ONE disk is the Monitor's working device, the device theOS can access Monitor files from. OSCONFIG also allows you to emulate a smaller machine.

The devices OSCONFIG recognizes are Profile, Corvus, and printers. If a device isn't named in the configuration program, the OS doesn't see it even if you explicitly mount it. To change your configuration, rerun OSCONFIG under the Monitor, FTP the new configuration file into the OS as SYSTEM.CONFIG, shutdown the OS, physically switch to the new arrangement, and then reboot the OS. If your boot volume contains no SYSTEM.CONFIG file, the only device configured is the disk you are booting.

OS VOLUME TYPES

The OS currently supports two types of file system volumes, one built on top of the Monitor's concept of logical volumes and one entirely independent of Monitor volumes. The type of OS file system volume built within a Monitor logical volume is what you've used for the last few months when running the OS under the Monitor on a single disk.

Under the current OS, you can only access this type of volume on the designated Monitor working device. This type of OS volume CANNOT be a boot volume. It can reside anywhere on the disk and its access is totally protected by the Monitor's mount table.

'OS Devices' is the term used to describe the second type of volume. This type of volume CAN be a boot volume. However, an OS device has only a single OS volume that must start at the beginning of the device. When you initialize that volume, you specify how many blocks (pages) are it has.

The OS initializes the specified number of pages on the OS device. For example, if you specify 9720 blocks when initializing a Profile as an OS device, all 9720 blocks are rewritten from the front of the disk without regard for any Monitor volumes that already exist there. The OS doesn't check the mount table to avoid destruction of existing Monitor volumes.

However, it is possible for a device to be both an OS and a Monitor device if you create the Monitor volumes BEYOND the portion of the disk used as the OS volume. To reserve room for the OS volume, create a Monitor volume (under the volume manager) starting at the first physical block (block 8) of the disk that has the same size as the number of blocks you need for the OS device. The remaining space is usable for other Monitor volumes.

WARNING: When initializing an OS device that is split between an OS volume and Monitor volumes, be sure to initialize the correct number of pages. Specifying too large a number of blocks results in the destruction of Monitor volumes that follow the OS volume. In general, BE CAREFUL when mixing OS volumes and Monitor volumes on one device.

SYSTEM FILES

The standalone OS is distributed on a Profile that holds a bootable copy of the OS, miscellaneous release files, and the installation utilities. The files below define the release. All files listed, except OSCONFIG and OSINSTALL, should be on the OS boot volume. All are required to boot and run the OS except for RS232TEST.

SYSTEM.OS - The main portion of the OS code.

SYSTEM.SHELL - OS command shell

SYSTEM.BT_PROF - The profile version of the OS loader

SYSTEM.BT_TWIG - The Twiggy version of the OS loader

SYSTEM.DEBUG - First part of Lisabug

SYSTEM.DEBUG2 - Second part of Lisabug

SYSTEM.LLD - Low level drivers

SYSTEM.CONFIG - User-produced definition of desired configuration

IOSPASLIB.OBJ - System runtime library

INTRINSIC.LIB - Intrinsic unit directory

RS232TEST - RS232 test program (optional)

OSINSTALL.TEXT - Exec file that transfers files onto your OS volume

OSCONFIG - Monitor-based utility to generate a SYSTEM.CONFIG file

The files in the following list may be useful to you, but are not required to install the OS.

SYSCALL.OBJ - Public system calls unit

PSYSCALL.OBJ - Privileged system calls unit

INSTALLING THE OS

The following are the steps required to install OS 5.2 onto a Profile:

1) Attach the library OS Profile to &3, the parallel port of your system, attach your target OS boot Profile to &5, the port on the N-port card that is second from the bottom, and attach the source of the Monitor's root volume (ie, #5:) at &4, the bottom port of the N-port card.

2) Boot the Monitor and make sure that the library Profile containing this version of the OS contains all the files listed above.

3) Be sure the Sysmgr 'Zero' has been run at some time on your target OS boot Profile. OS devices need to have a valid volume table, both to avoid confusion when examining the drive from the Monitor, and to allow the OS to use the drive as the Monitor's working device if necessary.

4) If you are developing programs on the Monitor to run on the OS, you have to transfer the following files from the library profile to a Monitor volume:

```
OS52:IOSPASLIB.OBJ
OS52:INTRINSIC.LIB
OS52:SYSCALL.OBJ
OS52:PSYSCALL.OBJ
```

Also transfer

```
OS52:OSCONFIG
```

to a Monitor volume so that you can change configurations later.

5) Run the OS52:OSCONFIG program to describe the configuration you want. The program expects single character numeric input. Save the configuration file in OS52:SYSTEM.CONFIG.

6) Boot the library copy of the OS (see the section below on booting the OS for directions).

7) Type 'P' to run the privileged filer and then type 'I' (for I(nit) to initialize your OS device (see the section on P(rivfiler if you need assistance). Use the name SLOT2CHAN2 to identify your profile at position &5. Depending on whether you have any Monitor volumes on your profile, you may want to respond with less than the maximum device size (9720) when I(nit asks how many pages (blocks) it should initialize for the OS volume. The OS uses 1300 blocks of your boot volume. So you will want to create at least 1500 blocks even if you don't intend to store anything else on the volume. Note that the OS boot volume must have enough space for the preallocated swap region, swap space for the applications, and swap space for data segments. Therefore, the minimum recommended size is 2000 blocks.

8) Type 'Q' to return to the OS Shell. Then type 'F' to run the OS Filer. Once you are in the filer, type 'M' to mount your newly initialized profile using the name SLOT2CHAN2. The Mount command is described under the OS Filer.

9) Type 'W' to change your working directory to the name of your OS volume. If you don't change the working directory, the macro that transfers the system files to your OS boot volume will not run correctly.

10) Type 'Q' to return to the OS Shell. Once you are in the Shell, type 'X' to execute <OS52:OSINSTALL, a macro that transfers each of the following files into your initialized volume using the FTP utility 'T(rans':

```
OS52:SYSTEM.OS
OS52:SYSTEM.SHELL
OS52:SYSTEM.BT_PROF
OS52:SYSTEM.BT_TWIG
OS52:SYSTEM.DEBUG
OS52:SYSTEM.DEBUG2
OS52:SYSTEM.LLD
OS52:SYSTEM.CONFIG
OS52:IOSPASLIB.OBJ
OS52:INTRINSIC.LIB
OS52:RS232TEST
```

11) Quit the OS and wait for the system to reset itself. Then, detach the library Profile that contains the OS, and detach your Profile that contains the OS from the Lisa. Reconnect your OS standalone Profile at the parallel port, and attach all other devices comprising the configuration you stored in the configuration file. You should now be able to boot the OS from your Profile (see BOOTING below for instructions).

Remember that the OS and UCSD file systems are not compatible.

HOW TO BOOT THE OS

The boot prom can boot either the OS or the Monitor. To decide which system to boot and which device to boot from, the prom selects the FIRST of the following list of possibilities that it encounters:

1. If one of the combinations of keys listed below is pressed at the right time, the prom selects the corresponding system/boot device.

   'Command' followed by 'h' means boot the OS from the Profile on the parallel port

   'Command' followed by 'f' means boot the OS from the top Twiggy drive

   'Command' followed by 'g' means boot the OS from the bottom Twiggy drive

   'Command' followed by 'm' means boot the Monitor from the Apple (for PROM 22)

   'Command' followed by 'a' means boot the Monitor from the Apple (for later PROMs)

   Learning the key press timing can be frustrating. The sweep pattern that appears about 3 or 4 seconds into the system power on process is your cue. Press and HOLD DOWN the command key after the sweep pattern appears, and then press the second key. You will probably make more mistakes by typing too soon than too late, so take your time. Within another 3 or 4 seconds either the 'BOOTING' message appears on the left of the screen (the prom saw the keys and is obeying) or the standard prom version display is seen (you'll have to try again). To try again, press the 'reset' button on the back of your machine, if you have one, or power your system off and back on. Pause at least 1 second between turning the machine on (in back) and pushing the 'power' button (in front).

   Version 102 or later of the boot prom makes a soft click instead of displaying the sweeping pattern when it's ready for you to type a boot device keycode, and a second click when it's no longer receptive.

2. If parameter memory is 'valid', the prom uses the boot device stored there. Only version 102 of the prom stores a valid combination of boot keys in parameter memory. No other method of writing to parameter memory exists yet.

3. Boot from the 'default' device. Currently, this means to boot the Monitor from the Apple. Some day, it will mean the top Twiggy.

The Profile must be left on for each attempt to boot the OS. Hopefully, this won't endanger disk integrity. If you have a Corvus attached to the system, you may want to turn it off before powering the system off and on.

After booting the OS, the Soroc displays the OS version number, the devices in the current configuration, and the numbers of the available volumes. REMEMBER: only one Monitor file disk is accessible.

SHUTTING DOWN THE OS

Whenever a user process returns to the Shell, you can quit the OS. However, if a user-process exception or system exception occurs, special action is necessary to preserve the integrity of files. During the normal course of running the OS, user and system data destined for a disk volume may still be in memory. If you reset the machine and reboot while data is in the buffer, the disk will be in an inconsistent state. The table below describes several situations that cause this problem and recommends an action for each.

Error	Action
Exception in USER process such as divide by zero, bus error, address error, etc.	Type 'g' from the debugger and the OS continues to abort the process and do any necessary clean up work.
NMI in USER process that is indicated by entering debugger in a domain other than zero AND without the debugger condition 'DOMAIN=2, OVERIDDEN TO 0'	Type 'g' from the debugger to continue executing the process. To abort the process, induce an artificial exception. One way to do this is to set PC to 0 ('pc 0') and then type 'g'. The process will probably get an illegal instruction exception and the OS should be able to abort it and do any clean up work necessary. REMEMBER: this only works if the domain IS NOT ZERO.
Exception in system code	Once in the debugger, type 'OSQUIT' from the debugger and the OS attempts to shut down the OS file system in an orderly fashion. You might have to type 'OSQUIT' several times before it works. DO NOT use NMI and 'rb' to reset the machine unless OSQUIT does not work after repeated attempts.

NMI in system code	Type 'g' to continue. To recover from a fatal error in the OS, type 'OSQUIT'. You may have to type 'OSQUIT' several times before it works DO NOT use NMI and 'rb' to reset the machine unless OSQUIT does not work after a dozen or so attempts.

PROGRAM DEVELOPMENT

To write a program that can run on the OS:

1. On the Monitor:

   Compile your program using the SYSCALL unit

   Link the compiled version of your program with IOSPASLIB

2. Boot the OS

3. T(ransfer the linked .OBJ file to an OS file system volume

4. X(ecute the program

THE OS COMMAND SHELL

When the OS comes up, a system process (the Root process) looks on the OS volume for a program file named SYSTEM.SHELL. If the OS finds one, it uses it as the OS command shell. If the OS doesn't find a SYSTEM.SHELL file, the Root process complains and goes automatically to the file transfer utility. At this point you can transfer any file from the UCSD world to serve as the OS shell. When you leave the file transfer utility, the Root process again looks for SYSTEM.SHELL. It repeats this cycle until it finally finds and starts up a shell.

To change the shell, kill the current SYSTEM.SHELL, transfer a new SYSTEM.SHELL to the OS volume, and then type 'Q' to quit. The Q(uit command asks whether or not you want to restart the Shell. If you type 'n' or press the carriage return, the OS terminates the Shell and shuts down the system. If you type 'y' in response to the question, the OS recreates the Shell from SYSTEM.SHELL. This procedure assumes that your current shell can kill and transfer files.

The position of a device determines its OS device name. The definitions of OS device names are as follows:

-PARAPORT is the device attached to the parallel port.

-SLOTxCHANy is the device attached to a 4-port card's slot x and channel y. Slots and channels are numbered 1, 2, and 3. Slot 1 is the slot furthest from the power supply side of the machine; channel 1 is the bottom channel. EXAMPLE: A drive connected to the bottom port on a 4-port card that is in slot 2 is mounted as device -SLOT2CHAN1.

The remainder of this section presents the OS Command Shell line and explains the OS command shell options. The OS command shell behaves like the UCSD command shell; to invoke an action, type the first character of the option you desire.

```
lisaOS: X(ecute, D(ebug, F(iler, P(rivFiler, T(ime, V(ers, O(ff, Q(uit
```

X(ecute	Executes a program. It prompts for the name of the program file to execute and expects the full OS file system name of a file that is on the OS volume. You must compile a program that runs on the OS with the SYSCALL unit and link it with IOSPASLIB before transferring it to the OS file system. If the character '<' precedes the file name, the Shell assumes that the file is an exec file. OS exec file format is the same as the Monitor format. The system determines which type of volume an exec file resides on by the file name used (see the T(rans command).

D(ebug	Debugs a program. D(ebug acts the same as the X(ecute command except that the program comes up with a breakpoint at its first instruction. Due to the breakpoint, the system enters Lisabug and you can debug as usual. After prompting for the name of the program to debug, the command asks if you also want to debug all the son processes. If you type 'n' for no, only the process created to run the program comes up with the breakpoint at the first instruction. If you type 'y' for yes, every process created until the main program terminates comes up with a breakpoint on the first instruction.
F(iler	Enters the Filer (described below).
P(rivFiler	Enters the privileged Filer (described below).
T(ime	Displays the current date/time setting and lets you enter a new date and/or time if desired. Type <CR> to indicate no change. To change the date or time, enter the new date and/or time in the format that the prompt specifies.
L(ib	Re-installs the Intrinsic Unit Directory file in memory. The command assumes that the new INTRINSIC.LIB file is already on the OS volume and that the Shell is the only process running in the system. If any error occurs during directory installation, a system error results and you must restart the OS. You can transfer and use a new INTRINSIC.LIB and use it while the OS is rebooting. Usually, no problems should occur when installing a new directory. NOTE: you cannot change IOSPASLIB using this command; you must reboot the OS to change it.
V(ers	Lists module version numbers and the OS release number. The OS group uses it to determine which versions of the OS components are being used.
O(ff	Turns Lisa off. The user is warned that power is about to be turned off. Answering yes ('y' or 'Y') to the warning prompt terminates the Shell and turns off the Lisa. Any other answer returns to the Shell command line.

Q(uit — Terminates the current Shell process. The user is asked if a new shell should be created or if the Operating System should be shut down and the Lisa reset (the power is left on). Note that the Q(uit and O(ff commands are

THE ACCEPTABLE WAYS TO LEAVE THE OS
AND RETURN TO THE MONITOR.

These alternatives allow the Operating System to completely close and flush files that are open and to put the disk in a consistent state. If you do not wish to shut down the the system, the OS tries to start another SYSTEM.SHELL program. Use this to change Shells while running under the OS. You can also type 'OSQUIT' to return to the Monitor. This alternative is not desirable but is encouraged if the other alternatives don't work.

THE FILER

There are two 'Filers' in the OS environment. The 'Filer' handles normal file operations. The 'PrivFiler' handles special privileged operations mostly used to manage volumes.

When prompted for a device name, a response of <CR> is sufficient to specify the current working directory. In general, however, a response of <CR> to a prompt indicates that the command should be aborted. In those situations where <CR> means the current working directory, a response of <ESC> aborts the command.

The first half of the Filer command line is:

Filer: T(rans, L(ist, N(ew, K(ill, R(ename, M(ount, U(nmount, Q(uit, ?

Note that '?' is a command, not a request for information. It causes the command prompt to flip to the other half of the command line and display the other available commands. The other half of the Filer command line is:

W(orkingDir, S(afety, D(eleteFiles

T(rans

T(rans invokes the file transfer utility FTP. FTP transfers files from the Monitor to the OS and from OS files to other OS files. Give FTP the source file name using either the UCSD file name syntax or the OS file name syntax, depending on the file, and the destination file name using the OS syntax. If a file with the destination file name already exists, FTP asks you for confirmation before writing over the old file. Once the transfer is complete, FTP asks for the next file to transfer. Type <cr> to exit.

Because two different file naming conventions are in use here, perhaps an example will be useful:

```
T(ransfer
What Lisa or UCSD file to transfer? VOL:MYTEXT.TEXT <cr>
What Lisa file to transfer into? -DISK-MYFILE <cr>
What UCSD file to transfer? <cr>
```

This example takes the Pascal text file MYTEXT.TEXT from the Pascal volume named VOL and places it in the Lisa file MYFILE that is on the Lisa volume named DISK.

Note that Paslib now supports RESET/REWRITE to an OS file. Therefore the prompt for the name of the file to transfer accepts either an OS file name or a Monitor file name.

Because Paslib supports both Monitor volumes and OS volumes, the following naming convention has been adopted in order to distinguish one type from the other. A file name represents a Monitor file only if a colon (:) is one of the characters in the name AND the leading character of the name is not a dash (-). So, for example, you can transfer an OS file named VOL:MYTEXT.TEXT as long as you add a dash to the beginning of the name. Note that even when you transfer a Monitor file that resides on the boot volume, you must type the volume number or name and the file name; typing only the file name is not sufficient to identify the file.

T(rans currently can only transfer Monitor files that reside in one of the SHORT directory volumes on the Monitor working device. The command considers the large directory volumes on the Monitor device invalid and skips over them when searching the volume for the specified file. If you want to transfer a file that is currently in a large directory, create a short directory volume and transfer the necessary file or files to that directory while running under the Monitor. Then, boot the OS and transfer the files. The next release of the OS or Paslib will support large directory Monitor volumes.

If you have multiple hard disks connected to your system via the 4-port card, you can only transfer the UCSD files stored on a single device. To select a working device you run the OSCONFIG program and copy the result into your OS boot volume.

Note that the transfer utility does not recognize the new Monitor file name syntax (DEV/VOL: FILE).

If you transfer a file into the Lisa file INTRINSIC.LIB, the system asks you if it should install the new Intrinsic Unit Directory immediately. The system installs it if you respond 'Y' or 'y'. If you choose not to install the new directory at that time, you must use the L(ib command later to install it yourself before running any programs that use the new INTRINSIC.LIB file.

L(ist

Lists the files on a given directory, their sizes and the disk space that each uses. The disk space size is the number of blocks (512 bytes) currently allocated to the file (the PEOF), whereas the file size is the number of bytes of data in the file (the LEOF).

N(ew

Creates a new file.

K(ill

Deletes a file.

R(ename

Renames an existing file or volume. If a volume is renamed, you must precede the volume name with a dash. Do not specify the dash if you are renaming a file on the working directory.

M(ount
U(nmount

The two commands permit you to manage multiple OS file system volumes.

S(afety

Toggles the safety switch of a file on or off. The command asks for a file name and then asks whether the switch should be turned on (respond 'y' to the question) or off (respond 'n' or just <CR>).

W(orkingDir

Displays the current working directory and then prompts for a new one. To change it, type the name of the new working directory; <CR> indicates no change. When changing the working directory, use a complete volume name (remember to include the '-') or the command has no effect. '-DEV9' and '-MyVol' are two example volume names. Once a working directory is set, partially specified pathnames are evaluated using that directory. If you UNMOUNT the volume containing the current working directory, the boot volume becomes the working directory.

D(eleteFiles

Deletes files using a simple wild card mechanism. The command first asks for the name of the directory to be searched and then asks for the partial file name for the search. The partial file name is the initial characters of the file names you want. For example, if you type 'ABC', the Filer searches for any file beginning with 'ABC'. If you type <cr>, all files in the directory match. After the directory search, D(eleteFiles prompts you to enter whether or not you want to delete the files, if any, that match the partial name. Type <ESC> to stop file deletion before going through the whole directory.

THE PRIVILEGED FILER

The P(rivFiler command line is:

PrivFiler: O(nline, E(ject, F(ix, I(nit, Z(ap, N(ewTwig, W(riteBT, Q(uit, ?

As with the Filer, the ? command flips to the other half of the PrivFiler's command line which is:

D(ump

O(nline

Lists each currently mounted volume and the device it is mounted on. It also prints the name of the current working directory.

E(ject

Ejects a Twiggy disk from the specified device. Note that the button on a drive will not eject a disk; you must use the E(ject command. However, the command does not eject a disk that is not mounted.

F(ix

Invokes the Scavenger, the volume restoration utility program. It recovers lost space on a volume and repairs damaged file structures. You cannot run the Scavenger against the OS boot volume. If you attempt to scavenge the boot volume, the Scavenger returns error 1228.

I(nit

Creates an OS file system volume. The volume initialized must not be mounted. After you specify the device name (without the '-'), the Filer asks for the set up information it needs. If the device is a diskette (not a Profile or the network), the media is automatically formatted.

Although I(nit destroys the current volume contents, the Pascal directory, if it is a small one, is untouched. Thus, the Monitor can still read the volume. Once you have initialized the volume, remember to mount it so you can use it. Boot tracks are automatically written to any initialized device as long as it is not the boot device.

DO NOT attempt to Initialize an illegal device.

Z(ap

Invalidates an OS file system volume. To use the volume again, you have to initialize the volume the next time you start up the OS. If you change your mind after Zapping a volume, just Zap it again. Zap makes the volume appear to be an unmountable non-OS volume. The Z(ero command in the Monitor is not equivalent to Zap.

N(ewTwig

Formats a Twiggy diskette. The command prompts for the device name; "UPPER" or "LOWER" are appropriate names for Twiggies. After formatting the diskette, you should initialize it as an OS volume. Note that you are no longer have to format a diskette before you initialize it.

W(riteBT

Writes boot track information on an initialized Twiggy diskette or Profile to allow you to boot the standalone OS. NOTE: you can't write boot tracks on your boot volume. Instead, boot the OS from another Profile or Twiggy, attach your boot Profile to the N-port card, and then write boot tracks to it. Users following the standard OS installation procedures should not end up with a boot volume without boot tracks. Because boot tracks are automatically written when a volume is initialized, this function is of minor usefulness outside the OS group.

D(ump

Provides a nicely formatted hexadecimal and ASCII dump of any page in the Lisa file system. The OS group uses D(ump primarily as a debugging aid.

Q(uit

Quit exits the PrivFiler and returns you to the OS command shell.

THE ASYNCHRONOUS FILE SYSTEM

Because your OS volume can only be on a Profile or a Twiggy, the OS blocks a process calling a system procedure that involves an I/O operation until the operation is complete. If there is a ready process at that time, the scheduler starts that process running during the time necessary for the I/O operation.

This feature may improve overall performance of the OS. However, it can cause some problems. It is possible with this feature that writeln messages from several processes can get interspersed. This occurs if a writeln message from one process interrupts a writeln message from another process currently blocked for an I/O operation.

Although this feature should not affect application programs, problems may occur with executing processes that share variables. A situation that could cause problems with shared data is the following. A process sets up a shared data address and then calls READ_DATA to this address. The READ_DATA call blocks this process and allows a second process, possibly of lower priority, to run. If the second process attempts to use the shared data, it might receive erroneous data. If you have any problems protecting shared data, consult the OS group.

OS PERFORMANCE

This section explains the file system improvements that are part of the 5.2 release of the Operating System and suggests how you can take advantage of them to improve program performance. Optimizations occurred in the following areas:

- Creation and killing of objects
- Allocation and deallocation of space
- Multiple-block I/O

Reorganizing the catalogs speeded up creating and killing objects. The OS now uses a hashing technique to map an object name to a location within a catalog. Other internal reorganizations further reduce the I/O time required for most catalog accesses.

In order to accomplish these changes in the 5.2 release time frame, the structure of a catalog entry remains the same. However, a catalog now has a fixed size without linked entries. The larger fixed-size catalog causes the only known performance degradation: a significant slowdown of Get_Next_Entry for catalogs that are nearly empty. Hopefully, this problem will be solved by the next OS release.

Space allocation and deallocation no longer write page label (hint) information. The write operation is postponed until data is written to the file because I/O has to occur at that time anyway. The implications of this change are:

- Although allocation and deallocation are significantly faster, reliability may be somewhat reduced. It is very important to use the recommended shutdown techniques discussed earlier to guarantee that the file system information is flushed to disk in a consistent state.

- Because automatic allocation on writes is still implemented, explicit allocation is useful mainly for guaranteeing the availability of disk space rather than for performance benefits.

Major optimizations were made to take advantage of any opportunity to do multiple-block I/O. The larger the request, the better the increase in speed is likely to be. Conversely, you will probably see no significant improvement in I/O operations that deal with only one byte at a time. However, there should be some improvement of speed in such I/O operations due to the fact that automatic write through to the disk no longer occurs.

To summarize, create and kill are much faster, allocate and deallocate, although much faster, are now useful mainly for managing availability of disk space, and reading and writing multiple blocks of data at a time is much faster. It is hard to state how much faster these operations are; feedback from user tests would be much appreciated. However, please see the Operating System group if you don't notice significant improvement in the above areas.

PRINTERS AND RS-232 INPUT/OUTPUT

The Operating System supports the parallel ports and one serial RS-232 port; the other RS-232 port is reserved for Lisabug on the standalone OS. The parallel ports on the 4-port card are named -slotxchany-anything, where x and y are numbers 1 through three depending on the configuration. The device pathname for the OS supported RS-232 port is '-RS232B-anything' where 'anything' is any sequence of characters. RS232B is the leftmost port when facing the front of the machine. There is no device control required for printing on the parallel ports. The remainder of this section is devoted to serial printing.

Follow the directions in this paragraph to set up a printer. Set the printer to handle 1200 baud serial communications. Connect the printer cable to a modem eliminator, and connect the modem eliminator to the RS232B port. If you want to connect the printer to a Soroc instead, set the Soroc to 1200 baud (set its rotary switch to 6) and connect the Soroc to the RS232B port using a standard Lisa-to-Soroc cable.

The default configuration is no parity, DTR handshake, 1200 Baud. You can change the configuration by using the DEVICE_CONTROL procedure. A sample program fragment that calls DEVICE_CONTROL follows.

```
VAR
    cparm: dctype;
    errnum: integer;
    path: pathname;

BEGIN
path:='-RS232B';
cparm.dcversion:=2;          (* note version change *)
cparm.dccode:=     << w >>; (* see below *)
cparm.dcdata[0]:=  << x >>;
cparm.dcdata[1]:=  << y >>;
cparm.dcdata[2]:=  << z >>;
DEVICE_CONTROL(errnum,path,cparm);
END;
```

<< w >>, << x >>, << y >>, and << z >> are defined as follows:

FUNCTION	<< w >>	<< x >>	<< y >>	<< z >>
Group A--Parity:				
No parity	1	0	--	--
Odd parity, no input parity checking	1	1	--	--
Odd parity	1	2	--	--
Even parity, no input parity checking	1	3	--	--
Even parity	1	4	--	--
Group B--Output Handshake:				
DTR handshake	2	--	--	--
XON/XOFF handshake	3	--	--	--
delay after Cr, LF	4	ms delay	--	--
Group C--Baud rate:	5	baud	--	--

FUNCTION	<< w >>	<< x >>	<< y >>	<< z >>
Group D--Input waiting:				
wait for full line	6	0	--	--
return whatever rec'd	6	1	--	--
Group E--Input handshake:				
no handshake	7	0	--	--
	9	-1	-1	65
DTR handshake	7	--	--	--
XON/XOFF handshake	8	--	--	--
Group F--Input type-ahead buffer:				
flush only	9	-1	-2	-2
flush & re-size	9	bytes	-2	-2
flush, re-size, and set thresh	9	bytes	low	hi
Group G--Disconnect Detection:				
none	10	0	0	--
device on RS232B	10	0	-128	--

To change the configuration, call DEVICE_CONTROL for the option you want in each group. You can set baud to any standard rate. However, 3600, 7200, and 19200 baud are available only on the RS232B port.

'Low' and 'Hi' under Group F set the low and high threshhold in the type ahead input buffer. When 'hi' or more bytes are in the input buffer, XOFF is sent or DTR is dropped. Then when 'Low' or fewer bytes are in the type ahead buffer, XON is sent or DTR is re-asserted. The size of the type ahead buffer can be anywhere between 0 and 64 bytes inclusive.

Once the device is properly configured, OPEN a pathname 'RS232B-anything' where 'anything' can be any string of characters. You can now WRITE_DATA and READ_DATA with any size data block to the refnum opened.

STACK SIZE

The stack size that a process requires depends on several factors. These include the amount of storage necessary for program global variables, regular unit global variables and intrinsic unit global variables, but do not include shared intrinsic variables.

Besides the static stack space requirements, a process also requires stack space dynamically for procedure stack frames. These stack frames contain the procedure linkage information, procedure local variables, and space for temporary expressions. The initial amount of dynamic stack space is obtained from the program file the process is to execute and is allocated when the OS creates a process. The default initial dynamic stack size is 10K (set by the Linker). The user can set the initial dynamic stack size to any desired value using the +S option of the Linker.

During the course of execution, it is possible for a program to require more dynamic stack space than is currently allocated to the stack (stack overflow). When this occurs, the operating system automatically expands the stack by the necessary amount. Stack expansions occur as needed until an expansion would make the stack larger than the maximum stack size contained in the program file. The default value for maximum stack size is 128K (again set by the Linker). You can set the maximum stack size to any desired value using the +T option of the Linker.

Under the current system, if a process requires a stack expansion that would cause the stack to exceed the maximum stack size, the process gets a bus error and enters LisaBug. Once in LisaBug, the system displays the bus error message and allows the user to do any debugging desired. To continue, type 'g' to exit LisaBug and allow the OS to abort the process.

Under the final (production) system, the Operating System terminates a process needing more stack space than the maximum. The cause of the termination, located in the exception information block associated with the SYS_TERMINATE exception, will indicate 'stk_overflow' (see Unit Syscall).

Currently, the Operating System does not allow a stack size greater than 128K (the size of a hardware segment). So if you specify a value greater than 128K in either the +S or +T option, the OS lowers it to 128K when the process is created. Note also that there can be a performance penalty associated with stack expansion since Memory Manager must be run in order to make space (possibly causing I/O) for the larger stack segment.

INTRINSIC UNITS

To use Intrinsic Units under the OS you need the Monitor release 8.0 versions of the compiler and code generator, the 8.2 versions of the Intrinsic Unit Manager and Intrinsic Unit Linker, an INTRINSIC.LIB file, and the linked library file IOSPASLIB.OBJ found on the OS release disks.

The INTRINSIC.LIB file used must contain the 4 units that comprise PasLib. These are units 1 (PASLIB), 102 (BLKIOINT), 103 (BLOCKIO), and 104 (PASHEAP). The INTRINSIC.LIB file may contain anything else that you require for the application. Before using the INTRINSIC.LIB and IOSPASLIB.OBJ to link a new unit or program, you must I(nstall the IOSPASLIB.OBJ from the OS release disk with the Intrinsic Unit Manager.

The INTRINSIC.LIB file, IOSPASLIB.OBJ file, and any other library files required must be on the Monitor root volume and the OS volume before executing programs under the OS.

You must compile programs that call OS routines using the SYSCALL unit. If a program calls anything from the privileged OS interface, you must include the PSYSCALL unit as well. In addition, you must link programs calling OS routines from either interface with IOSPASLIB.OBJ.

Because both the INTRINSIC.LIB file and the various library files are required to run any programs that use Intrinsic Units, several problems can occur if you are not careful about keeping these files consistent with each other. If a library file is ever changed, you must re-install it in INTRINSIC.LIB, and you must transfer both the new library file and the new INTRINSIC.LIB to the OS volume.

When you transfer a new INTRINSIC.LIB file to the OS volume, you must also change the memory resident copy of INTRINSIC.LIB. You can change the memory resident copy of the file either while in the T(ransfer command of the F(iler or later with the L(ib command of the Shell (see the descriptions of these commands for details).

If any of these steps are omitted, various errors can occur. For example, if you define a new Intrinsic Unit, build a program that uses the unit, but forget to transfer and change the INTRINSIC.LIB file on the OS volume, Make_Process returns an error saying that the unit was not found in the Intrinsic Unit Directory. The error occurs because it is not in the memory copy of INTRINSIC.LIB.

As an aid in tracking these kinds of errors, the OS Loader currently displays the Intrinsic Unit number and name that was not found on the screen. This display will not be in the production system. Similar errors occur when you change the name or type of a unit and forget to transfer over the new INTRINSIC.LIB and/or library file before executing a program that uses the unit.

More complicated errors can occur if the size of a shared code segment associated with an Intrinsic Unit or its location in a library file changes and the new INTRINSIC.LIB and/or library file is not transferred to the OS volume. In this case, the error is not detected until the code segment is swapped into memory. At this point, you get the message

*** Error swapping in private code segment # nn for process id # pp
OR
*** Error swapping in shared code sement # nn (segname) for process id # pp

where nn is the code segment number the application process uses, segname is the name of the shared segment from Intrinsic.Lib, and pp is the process identification number of the process for whom the segment is swapped in.

If the swap-in error is for a shared segment, it is generally due to an inconsistency between Intrinsic.Lib and the library file containing the shared segment. If this is the case, the correct Intrinsic.Lib and the library file associated with the bad segment are probably not on the OS volume.

If the swap-in error is for a private segment, it is generally due to either an improper link or a bad spot on the disk. To solve this problem, relink the program and transfer the relinked version to the OS volume.

Regardless of the kind of swap-in error, type < ret > to continue. The OS terminates the failing process and the information block associated with the process's SYS_TERMINATE exception indicates that the OS is terminating the process due to a swap-in error.

PASLIB

The standalone OS does not support some of the Paslib routines. The remainder of this section explains how you use PASLIB routines in the OS world. If an unsupported function is called in the standalone OS, the system displays the following message:

MONITOR TRAP (E) occurred, index=<iiii> (routine name) in process of gid <gggg>

where <iiii> is the routine's index to the Monitor's TRAP E handler. See the Pascal Development System Internal Documentation for the identity of an index without a routine name.

The standalone OS does not support unit IO routines such as Unitread and Unitwrite and does not support the seek routine. However, it does support the GOTOXY routine.

The Paslib routines for value range check and string index check run in the OS environment. If the range check indicates an error in OS code, a system error is signalled. The message displayed is:

```
VALUE RANGE ERROR in system code!
value to check = <vvvv>  lower bound = <nnnn>  upper bound = <uuuu>
return pc = <pppppp>  caller a6 = <cccccc>
Going to Lisabug, type g to continue.
```

where:

<pppppp> is the address of the next statement of the call to the range check routine in Paslib,

<cccccc> is the address of the link field at the time of the call to paslib

or:

```
ILLEGAL STRING INDEX in system code!
value to check = <vvvv>  lower bound = <nnnn>  upper bound = <uuuu>
return pc = <pppppp>  caller a6 = <cccccc>
Going to Lisabug, type g to continue.
```

In this case and for other errors in system code, use the OSQUIT facility described earlier instead of typing g to continue. If you type g and try to continue, you get system error 10201 and you must reboot the system.

If a range check error occurs in application code, the system exception 'SYS_VALUE_OOB' is signalled. The message displayed is:

```
VALUE RANGE ERROR in process gid <gggg>
value to check = <vvvv>  lower bound = <nnnn>  upper bound = <uuuu>
return pc = <pppppp>  caller a6 = <cccccc>
Going to Lisabug, type g to continue.
```

or:

```
ILLEGAL STRING INDEX in process of gid <gggg>
value to check = <vvvv>  lower bound = <nnnn>  upper bound = <uuuu>
return pc = <pppppp>  caller a6 = <cccccc>
Going to Lisabug, type g to continue.
```

If the process has not declared an exception handler for the exception that occurs, the system exception handler is entered after you type 'g' to contine. It terminates the process. If the process has declared a handler, the handler is called after you type 'g', and the process then continues execution.

The intrinsic procedure HALT calls TERMINATE_PROCESS without passing an event.

The block IO routines, RESET, REWRITE, BLOCKREAD, BLOCKWRITE, and IORESULT, act in the operating system just as they do in the Monitor. In fact, RESET and REWRITE accept both Monitor and OS file names.

In order to distinguish between the two types of files in RESET and REWRITE, Paslib has adopted the following file naming convention: a file name designates a Monitor file only if the name includes a colon (:) AND does not begin with a dash (-). All other files are considered OS files.

You can override the file system mode with a call to SetMPrefix. This call allows you to choose the file system you want; its format is:

```
procedure SetMPrefix (var errnum: integer;
                          prefix: vid);
```

where vid is of type string [7]. If you set prefix to '%%OS', later calls to RESET or REWRITE assume that file names refer to OS files. If you set prefix to '%%MO', subsequent calls to the two routines assume that file names refer to Monitor files. The file naming conventions given above are in effect if you set prefix to '%%BO'. The file system mode you choose remains in effect until the next call to SetMPrefix.

Before calling SetMPrefix, you must declare it as external. This call will no longer be necessary once the entire development system is running on top of the OS and OS support of Monitor volumes isn't necessary.

The OS file system mode does not support the file name construct [X], where X is a number of blocks. There is no need for the OS to support this syntax because an OS file expands as needed, and will not take the largest available space if X is not specified. In fact, if you do append the construct to the file name, it becomes part of the file name.

IORESULT, in the OS file system mode, returns the same error values as Monitor files, but the error return numbers are in the OS file system range. The current mapping of OS to UCSD file system error values is:

OS Error Value	UCSD Error Value
882	8
894	6
921, 946	7
941	5
948	10

Other errors are not mapped. The above error mappings will no longer exist once the OS stops supporting Monitor files.

Note also that IORESULT now returns error 11 (duplicate file) as a warning (-11) and completes the operation as normal when the error results from a second REWRITE of a file without a CLOSE after the first REWRITE. This change allows the user to decide whether to continue the program if the error occurs because of a program crash or to stop execution if there is a name conflict with another program that has already done a REWRITE to the same file.

For Monitor files, only units 5, and 9 through 20 are considered block structured devices. Block IO to a non-block structured device is not supported. IORESULT can return an additional error number:

17 - device error, non-zero value returned from last LISAIO call

Text file block IO works as expected. RESET and REWRITE of a text file (.TEXT suffix) sets the current block number to 2, thereby bypassing the text file header blocks. Note that RESET and REWRITE only accept names of files on the working device. In addition, the two routines do not support the new Monitor file name syntax (DEV/VOL: FILE) yet.

THE PASCAL HEAP

The heap is one contiguous piece of memory. It obtains a contiguous piece of memory with a call to Make_Dataseg. The heap works automatically without any initialization call; there are default values for size of the heap, the LDSN used, etc.

Most users of the heap should use it the same way as they use the Monitor heap. For those having special needs, there is a call, PLINITHEAP, that specifies the LDSN, the initial heap SIZE, whether the heap can be swapped to disk, and the heap DELTA size. When you use PLINITHEAP, you must call it before making a call to the other heap routines.

```
PLINITHEAP ( ERROR, REFNUM, SIZE, DELTA, 9, FALSE);
```

PLINITHEAP is defined as follows in the PASLIBCALL unit:

```
procedure PLINITHEAP (var ERNUM, REFNUM: integer;
                          SIZE, DELTA: longint;
                          LDSN: integer;
                          SWAPABLE: boolean);
```

where ERNUM is the error number returned if the procedure has any problems making a data segment having a mem_size of SIZE bytes, LDSN is the LDSN used for the heap (the default is 5), REFNUM is the refnum of the heap, DELTA is the amount the data segment increases when the current space is used up (If you use a large heap, use a large number for DELTA), and SWAPABLE is the boolean that determines whether the system can swap the heap data segment out to disk if it needs to.

When a Pascal program starts execution, there is no heap space allocated (no data segment made). On the first call to one of the heap routines or on the first PLINITHEAP call, the heap is created with either a default size of 16k bytes or the size specified in the PLINITHEAP call.

PLINITHEAP makes the heap as a private data segment so that the OS will remove it when the process calling PLINITHEAP terminates. Note that when the heap is initialized, size and delta are put on 512 byte block boundries. Therefore, if you use the PLINITHEAP call and specify values for size and delta that don't fall on block boundaries, the procedure increases the values to the next block boundary.

If the heap runs out of space while it is being used, the size of the heap is increased by the default of 16k or the DELTA specified in PLINITHEAP. The default LDSN used is LDSN 5. If you want a different LDSN for the heap data segment, call PLINITHEAP. Remember that the size of a data segment is limited by the LDSN you use. For LDSN 16, you can only get 128k (actually 96k safely), for LDSN 15 256k, for LDSN 14 384k, ...

If SWAPABLE is true, the heap is made with disc_size equal to SIZE so the data segment will not be memory resident. The default for SWAPABLE is false. When SWAPABLE is false the procedure creates a data segment that has a disc_size of 0 which makes it memory resident.

The unit PASLIBCALL contains the interface for the PLINITHEAP call. PASLIBCALL will contain the interface to all procedures that effect the PASLIB. PASLIBCALL will be released with Tippe.

Currently, the OS supports the built in Pascal heap routines NEW, MEMAVAIL, MARK, and RELEASE.

If you call NEW and there is not enough space, the size of the heap is increased by the default of 16k or the deltasize specified in PLINITHEAP.

MEMAVAIL gives you the maximum number of words you could ever expect to get and takes into account the LDSN you used as well as the amount of free space the OS currently has available. If there is another process using memory concurrently, it's use of memory also effects MEMAVAIL. MEMAVAIL does not show the amount of memory left in the heap's data segment alone since the heap's data segment can grow and shrink over time.

MARK works as it does in the Monitor.

If you release the heap to a point within the original size of the heap data segment, the heap data segment is reduced to its original size. Other than this, RELEASE works as it does in the Monitor.

The current heap could be replaced later by heap routines within the standard storage manager. There are currently implementation dependencies in the Compiler, and probably other parts of the system, that require a UCSD style contiguous heap. If these dependencies are removed in time, we may go to a fancier heap for first release that contains DISPOSE and possibly other features.

THE OS INTERFACE

```
UNIT syscall;                    (* system call definitions unit *)
INTRINSIC;

INTERFACE

  CONST
    max_ename = 32;          (* maximum length of a file system object name *)
    len_exname = 16;         (* length of exception name *)
    size_exdata = 11;        (* 48 bytes, exception data block should have the same
                                size as r_eventblk, received event block *)

    size_etext = 9;          (* event text size - 40 bytes *)
    size_waitlist = 10;      (* size of wait list - should be same as reqptr_list *)

                                (* exception kind definitions for 'SYS_TERMINATE'
                                   exception   *)
    call_term = 0;              (* process called terminate_process *)
    ended     = 1;              (* process executed 'end' statement *)
    self_killed = 2;            (* process called kill_process on self *)
    killed      = 3;            (* process was killed by another process *)
    fthr_term   = 4;            (* process's father is terminating *)
    bad_syscall = 5;            (* process made invalid sys call - subcode bad *)
    bad_errnum  = 6;            (* process passed bad address for errnum parm *)
    swap_error  = 7;            (* process aborted due to code swap-in error *)
    stk_overflow = 8;           (* process exceeded max size (+T nnn) of stack *)
    data_overflow = 9;          (* process tried to exceed max data space size *)
    parity_err  = 10;           (* process got a parity error while executing *)

    def_div_zero   = 11;        (* default handler for div zero exception was called *)
    def_value_oob  = 12;        (* " for value oob exception *)
    def_ovfw       = 13;        (* " for overflow exception *)
    def_nmi_key    = 14;        (* " for NMI key exception *)
    def_range      = 15;        (* " for 'SYS_VALUE_OOB' excep due to value range err *)
    def_str_index  = 16;        (* " for 'SYS_VALUE_OOB' excep due to string index err*)

    bus_error = 21;             (* bus error occurred *)
    addr_error = 22;            (* address error occurred *)
    illg_inst = 23;             (* illegal instruction trap occurred *)
    priv_violation = 24;        (* privilege violation trap occurred *)
    line_1010 = 26;             (* line 1010 emulator occurred *)
    line_1111 = 27;             (* line 1111 emulator occurred *)

    div_zero    = 31;           (* exception kind definitions for hardware exception *)
    value_oob   = 32;
    ovfw        = 33;
    nmi_key     = 34;
    value_range = 35;           (* excep kind for value range and string index error *)
    str_index   = 36;           (* Note that these two cause 'SYS_VALUE_OOB' excep *)
```

```
TYPE

  pathname = string [255];
  e_name = string [max_ename];
  namestring = string [20];
  procinfoRec = record
                  progpathname : pathname;
                  global_id    : longint;
                  father_id    : longint;
                  priority     : 1..255;
                  state        : (pactive, psuspended, pwaiting);
                  data_in      : boolean
                end;


  Tdstype = (ds_shared, ds_private); (* types of data segments *)

  dsinfoRec = record
                mem_size : longint;
                disc_size: longint;
                numb_open : integer;
                ldsn : integer;
                boundF : boolean;
                presentF : boolean;
                creatorF : boolean;
                rwaccess : boolean;
                segptr : longint;
                volname: e_name;
              end;


  t_ex_name = string [len_exname];                    (* exception name *)
  longadr = ^longint;
  t_ex_state = (enabled, queued, ignored);            (* exception state *)
  p_ex_data = ^t_ex_data;
  t_ex_data = array [0..size_exdata] of longint;      (* exception data blk *)
  t_ex_sts = record                                   (* exception status *)
               ex_occurred_f : boolean;               (* exception occurred flag*)
               ex_state : t_ex_state;                 (* exception state *)
               num_excep : integer;                   (* number of exceptions q'ed*)
               hdl_adr : longadr;                     (* handler address *)
             end;
```

```
p_env_blk = ^env_blk;
env_blk = record                        (* environment block to pass to handler *)
            pc : longint;                   (* program counter *)
            sr : integer;                   (* status register *)
            d0 : longint;                   (* data registers 0 - 7 *)
            d1 : longint;
            d2 : longint;
            d3 : longint;
            d4 : longint;
            d5 : longint;
            d6 : longint;
            d7 : longint;
            a0 : longint;                   (* address registers 0 - 7 *)
            a1 : longint;
            a2 : longint;
            a3 : longint;
            a4 : longint;
            a5 : longint;
            a6 : longint;
            a7 : longint;
          end;

p_term_ex_data = ^term_ex_data;
term_ex_data = record                           (* terminate exception data block *)
                case excep_kind : longint of
                  call_term,
                  ended,
                  self_killed,
                  killed,
                  fthr_term,
                  bad_syscall,
                  bad_errnum,
                  swap_error,
                  stk_overflow,
                  data_overflow,
                  parity_err : ();              (* due to process termination *)

                  illg_inst,
                  priv_violation,           (* due to illegal instruction,
                                                        privilege violation *)
                  line_1010,
                  line_1111,                    (* due to line 1010, 1111 emulator *)
                  def_div_zero,
                  def_value_oob,
                  def_ovfw,
                  def_nmi_key                   (* terminate due to default handler for
                                                   hardware exception *)
                    : (sr : integer;
                       pc : longint);           (* at the time of occurrence *)
                  def_range,
```

```
                    def_str_index              (* terminate due to default handler for
                                                  'SYS_VALUE_OOB' excep for value
                                                  range or string index error *)
                      : (value_check : integer;
                         upper_bound : integer;
                         lower_bound : integer;
                         return_pc   : longint;
                         caller_a6   : longint);
                    bus_error,
                    addr_error                 (* due to bus error or address error *)
                      : (fun_field : packed record            (* one integer *)
                                       filler : 0..$7ff;      (* 11 bits *)
                                       r_w_flag : boolean;
                                       i_n_flag : boolean;
                                       fun_code : 0..7;       (* 3 bits *)
                                     end;
                         access_adr : longint;
                         inst_register : integer;
                         sr_error : integer;
                         pc_error : longint);
                end;

p_hard_ex_data = ^hard_ex_data;
hard_ex_data = record                  (* hardware exception data block *)
                 case excep_kind : longint of
                  div_zero, value_oob, ovfw
                    : (sr : integer;
                       pc : longint);
                  value_range, str_index
                    : (value_check : integer;
                       upper_bound : integer;
                       lower_bound : integer;
                       return_pc   : longint;
                       caller_a6   : longint);
               end;


accesses = (dread, dwrite, append, private, global_refnum);
mset = set of accesses;
iomode = (absolute, relative, sequential);

UID = record  (*unique id*)
  a,b: longint
end;

timestmp_interval = record             (* time interval *)
                      sec : longint;   (* number of seconds *)
                      msec : 0..999;   (* number of milliseconds within a second *)
                    end;
```

```
    info_type = (device_t, volume_t, object_t);
    devtype = (diskdev, pascalbd, seqdev, bitbkt, non_io);
    filetype = (undefined, MDDFfile, rootcat, freelist, badblocks,
                sysdata, spool, exec, usercat, pipe, bootfile,
                swapdata, swapcode, ramap, userfile, killedobject);

    entrytype= (emptyentry, catentry, linkentry, fileentry, pipeentry, ecentry,
                killedentry);

    fs_info = record
              name : e_name;
              devnum : integer;
              machine_id : longint;
              case otype : info_type of
                device_t, volume_t: (
                  iochannel : integer;
                  devt : devtype;
                  slot_no : integer;
                  fs_size : longint;
                  vol_size : longint;
                  blockstructured, mounted : boolean;
                  opencount : longint;
                  privatedev, remote, lockeddev : boolean;
                  mount_pending, unmount_pending : boolean;
                  volname, password : e_name;
                  fsversion, volnum : integer;
                  volid : UID;
                  blocksize, datasize, clustersize, filecount : integer;
                  freecount : longint;
                  DTVC, DTCC, DTVB, DTVS : longint;
                  master_copy_id, copy_thread : longint;
                  overmount_stamp : UID;
                  privileged, write_protected : boolean;
                  master, copy, scavenge_flag : boolean;
                  vol_left_mounted : boolean );

                object_t : (
                  size : longint;
                  psize : longint;   (* physical file size in bytes *)
                  lpsize : integer;  (* logical page size in bytes for this file *)
                  ftype : filetype;
                  etype : entrytype;
                  DTC, DTA, DTM, DTB, DTS : longint;
                  refnum : integer;
                  fmark : longint;
                  acmode : mset;
                  nreaders, nwriters, nusers : integer;
                  fuid : UID;
                  eof, safety_on, kswitch : boolean;
                  private, locked, protected, master_file : boolean;
                  file_scavenged, file_closed_by_OS, file_left_open : boolean )
              end;
```

```
dctype = record
           dcversion : integer;
           dccode : integer;
           dcdata : array [0..9] of longint; (* user/driver defined data *)
         end;

t_waitlist = record                          (* wait list *)
               length : integer;
               refnum : array [0..size_waitlist] of integer;
             end;

t_eheader = record                           (* event header *)
              send_pid : longint;            (* sender's process id *)
              event_type : longint;          (* type of event *)
            end;

t_event_text = array [0..size_etext] of longint;
p_r_eventblk = ^r_eventblk;
r_eventblk = record
               event_header : t_eheader;
               event_text : t_event_text;
             end;

p_s_eventblk = ^s_eventblk;
s_eventblk = t_event_text;

time_rec = record
             year : integer;
             day : 1..366;                   (* julian date *)
             hour : -23..23;
             minute : -59..59;
             second : 0..59;
             msec : 0..999;
           end;

chn_kind = (wait_ec, call_ec);
t_chn_sts = record                           (* channel status *)
              chn_type : chn_kind;           (* channel type *)
              num_events : integer;          (* number of events queued *)
              open_recv : integer;           (* number of opens for receiving *)
              open_send : integer;           (* number of opens for sending *)
              ec_name : pathname;            (* event channel name *)
            end;

hour_range = -23..23;
minute_range = -59..59;
```

```
(* File System calls *)

  procedure MAKE_FILE (var ecode:integer; var path:pathname; label_size:integer);

  procedure MAKE_PIPE (var ecode:integer; var path:pathname; label_size:integer);

  procedure MAKE_CATALOG (var ecode:integer; var path:pathname; label_size:integer);

  procedure MAKE_LINK (var ecode:integer; var path, ref:pathname; label_size:integer);

  procedure KILL_OBJECT (var ecode:integer; var path:pathname);

  procedure OPEN (var ecode:integer; var path:pathname; var refnum:integer; manip:mset)

  procedure CLOSE_OBJECT (var ecode:integer; refnum:integer);

  procedure READ_DATA (var ecode : integer;
                           refnum : integer;
                        data_addr : longint;
                            count : longint;
                       var actual : longint;
                             mode : iomode;
                           offset : longint);

  procedure WRITE_DATA (var ecode : integer;
                            refnum : integer;
                         data_addr : longint;
                             count : longint;
                        var actual : longint;
                              mode : iomode;
                            offset : longint);

  procedure FLUSH (var ecode:integer; refnum:integer);

  procedure LOOKUP (var ecode : integer;
                        var path : pathname;
                   var attributes : fs_info);

  procedure INFO (var ecode:integer; refnum:integer; var refinfo:fs_info);

  procedure ALLOCATE (var ecode : integer;
                          refnum : integer;
                       contiguous : boolean;
                            count : longint;
                       var actual : longint);

  procedure TRUNCATE (var ecode : integer;  refnum : integer);

  procedure COMPACT (var ecode : integer;  refnum : integer);

  procedure RENAME_ENTRY ( var ecode:integer; var path:pathname; var newname : e_name )
```

```
  procedure READ_LABEL (  var ecode : integer;
                           var path : pathname;
                         data_addr : longint;
                             count : longint;
                        var actual : longint  );

  procedure WRITE_LABEL (  var ecode : integer;
                           var path : pathname;
                         data_addr : longint;
                             count : longint;
                        var actual : longint  );

  procedure MOUNT ( var ecode:integer; var vname : e_name; var password : e_name ;
                    var devname : e_name);

  procedure UNMOUNT ( var ecode:integer; var vname : e_name );

  procedure SET_WORKING_DIR ( var ecode:integer; var path:pathname );

  procedure GET_WORKING_DIR ( var ecode:integer; var path:pathname );

  procedure SET_SAFETY ( var ecode:integer; var path:pathname; on_off:boolean );

  procedure DEVICE_CONTROL ( var ecode:integer; var path:pathname;
                             cparm : dctype );

  procedure RESET_CATALOG (var ecode : integer; var path : pathname);

  procedure GET_NEXT_ENTRY (var ecode : integer; var prefix, entry : e_name);

  procedure GET_DEV_NAME (var ecode : integer; var path : pathname;
                          var devname : e_name);

  procedure SET_FILE_INFO ( var ecode  : integer;
                                refnum : integer;
                                fsi    : fs_info );


(* Process Management system calls *)

  function My_ID : longint;

  procedure Info_Process (var errnum : integer;  proc_id : longint;
                          var proc_info : procinfoRec);

  procedure Yield_CPU (var errnum : integer;  to_any : boolean);

  procedure SetPriority_Process (var errnum : integer;  proc_id : longint;
                                 new_priority : integer);

  procedure Suspend_Process (var errnum : integer;  proc_id : longint;
                             susp_family : boolean);
```

```
  procedure Activate_Process (var errnum : integer;  proc_id : longint;
                              act_family : boolean);

  procedure Kill_Process (var errnum : integer;  proc_id : longint);

  procedure Terminate_Process (var errnum : integer;  event_ptr : p_s_eventblk);

  procedure Make_Process (var errnum : integer;  var proc_id : longint;
                          var progfile : pathname;  var entryname : namestring;
                          evnt_chn_refnum : integer);


(* Memory Management system calls *)

  procedure make_dataseg(var errnum: integer; var segname: pathname;
                         mem_size, disc_size: longint; var refnum: integer;
                         var segptr: longint; ldsn: integer; dstype: Tdstype);

  procedure kill_dataseg (var errnum : integer; var segname : pathname);

  procedure open_dataseg (var errnum : integer; var segname : pathname;
                          var refnum : integer; var segptr : longint;
                          ldsn : integer);

  procedure close_dataseg (var errnum : integer; refnum : integer);

  procedure size_dataseg (var errnum : integer; refnum : integer;
                          deltamemsize : longint; var newmemsize : longint;
                          deltadiscsize: longint; var newdiscsize: longint);

  procedure info_dataseg (var errnum : integer; refnum : integer;
                          var dsinfo : dsinfoRec);

  procedure setaccess_dataseg (var errnum : integer; refnum : integer;
                               readonly : boolean);

  procedure unbind_dataseg (var errnum : integer; refnum : integer);

  procedure bind_dataseg(var errnum : integer; refnum : integer);

  procedure info_ldsn (var errnum : integer; ldsn: integer; var refnum: integer);

  procedure flush_dataseg(var errnum: integer; refnum: integer);

  procedure MEM_INFO(var errnum: integer;
                     var swapspace, dataspace,
                         cur_codesize, max_codesize: longint);
```

```
(* Exception Management system calls *)

  procedure declare_excep_hdl (var errnum : integer;
                               var excep_name : t_ex_name;
                               entry_point : longadr);

  procedure disable_excep (var errnum : integer;
                           var excep_name : t_ex_name;
                           queue : boolean);

  procedure enable_excep (var errnum : integer;
                          var excep_name : t_ex_name);

  procedure signal_excep (var errnum : integer;
                          var excep_name : t_ex_name;
                          excep_data : t_ex_data);

  procedure info_excep (var errnum : integer;
                        var excep_name : t_ex_name;
                        var excep_status : t_ex_sts);

  procedure flush_excep (var errnum : integer;
                         var excep_name : t_ex_name);


(* Event Channel management system calls *)

  procedure make_event_chn (var errnum : integer;
                            var event_chn_name : pathname);

  procedure kill_event_chn (var errnum : integer;
                            var event_chn_name : pathname);

  procedure open_event_chn (var errnum : integer;
                            var event_chn_name : pathname;
                            var refnum : integer;
                            var excep_name : t_ex_name;
                            receiver : boolean);

  procedure close_event_chn (var errnum : integer;
                             refnum : integer);

  procedure info_event_chn (var errnum : integer;
                            refnum : integer;
                            var chn_info : t_chn_sts);

  procedure wait_event_chn (var errnum : integer;
                            var wait_list : t_waitlist;
                            var refnum : integer;
                            event_ptr : p_r_eventblk);
```

```
  procedure flush_event_chn (var errnum : integer;
                             refnum : integer);

  procedure send_event_chn (var errnum : integer;
                            refnum : integer;
                            event_ptr : p_s_eventblk;
                            interval : timestmp_interval;
                            clktime : time_rec);


(* Timer functions system calls *)

  procedure delay_time (var errnum : integer;
                        interval : timestmp_interval;
                        clktime : time_rec);

  procedure get_time (var errnum : integer;
                      var gmt_time : time_rec);

  procedure set_local_time_diff (var errnum : integer;
                                 hour : hour_range;
                                 minute : minute_range);

  procedure convert_time (var errnum : integer;
                          var gmt_time : time_rec;
                          var local_time : time_rec;
                          to_gmt : boolean);
```

```
UNIT psyscall;                    (* privileged system call definitions unit *)

INTRINSIC;

INTERFACE

  (*$U object:syscall.obj *)
  USES syscall;

const buff_too_small = 1158;
      e_sdubd = 1159;
      ddev_too_small = 1160;
      inv_shutdown_mode = 1161;
      pwr_already_off = 1162;
      badcmd_err = 1163;
      nottwig_err = 1164;
      notmounted_err = 1165;
      alreadymounted_err = 1166;
      notblockstr_err = 1167;

      maxILFile = 64;   { maximum intrinsic library file number }


  type
    vers_info = record  (* version information record *)
                  PPrim_V : integer;
                  PM_V :    integer;
                  GDV :     integer;
                  MMPrimV : integer;
                  MMV :     integer;
                  DSV :     integer;
                  ExprmV :  integer;
                  ExmgrV :  integer;
                  ECV :     integer;
                  TimeV :   integer;
                  VMV :     integer;
                  SFV :     integer;
                  PrimV :   integer;
                  UIV :     integer;
                  InitV :   integer;
                  CUR_V :   integer;
                  OSVers :  integer
                end;
```

```
ut_commands = (no_op,
               online,
               initvol,
               zap,
               dumpdata,
               setfstrace,
               fsscavenge,
               writeBT,
               format,
               verify,
               eject,
               flushbuffers,
               boot_unmount,
               boot_remount,
               copy_volume,
               shut_down_sys,
               mount_BD);

sm_type = (restart_shell, reset_machine, kill_power);

ScvOptions = ( no_abort, no_kill, no_pipes, no_discard );

ut_parmt = record
             gp_parm : longint;
             case command : ut_commands of

               no_op,
               online,
               flushbuffers,
               boot_remount : (                          );

               shut_down_sys: (sd_mode : sm_type         );

               initvol      : (idev_name : e_name;
                               pages : longint;
                               newvolname : e_name;
                               newpassword : e_name      );

               zap,
               writeBT,
               format,
               verify,
               eject        : (dev_name : e_name         );

               dumpdata     : (ddev_name : e_name;
                               pagenum : longint         );

               setfstrace,
               boot_unmount : (level : integer           );
```

```
                fsscavenge    : (sdev_name : e_name;
                                 soptions : ScvOptions      );

                copy_volume   : (from_dev : e_name;
                                 to_dev   : e_name;
                                 buffaddr : longint;
                                 buffsize : longint         );

                mount_BD      : (mon_unitnum : integer;
                                 twig_unitnum : integer     )  (* 1 = UPPER *)
                                                               (* 2 = LOWER *)
              end;


  ioop = (readop, writeop);

  refnum_type = (frefnum, dsrefnum, ecrefnum);

  openrec = record                                (* open list info record *)
              procid : longint;
              refnum : integer;
              refntype : refnum_type;
              globalrefn : boolean;
            end;

  Tlog_cmds = (log_dump,log_flush,log_reset,log_shutdown);(*logging commands*)


  { Intrinsic Library File # list }

  lFileRange = 1..maxILFile;
  lf_list = packed array[lFileRange] of boolean;   { list of library files used
                                                     by a program }


  { Lisa Office System parameter memory type }

  pmByteUniqueForAllTheDamnCryBabies = -128..127;
  pMemRec = array[1..62] of pmByteUniqueForAllTheDamnCryBabies;


 {configuration stuff: }

 tports = (seriala, serialb, parallel, slot11, slot12, slot1X,
           slot21, slot22, slot2X, slot31, slot32, slot3X);

 boot_tports = (uppertwig, lowertwig, pport, b_slot11, b_slot12, b_slot1X,
           b_slot21, b_slot22, b_slot2X, b_slot31, b_slot32, b_slot3X);
```

```
card_types = (no_card, apple_card, n_port_card, net_card, laser_card);

slot_array = array [1..3] of card_types;



procedure POPEN (var ecode : integer;
                   var path : pathname;
                 var refnum : integer;
                     manip : mset;
                var allowed : boolean  );

procedure protect (var ecode : integer;
                     var path : pathname;
                     ismaster : boolean;
                  m_serial_no : longint   );

procedure get_serial_no (var ecode : integer; var s_no : longint);

procedure GET_OPEN_LIST (var ecode : integer; var devname : e_name;
                           var openinfo : openrec);

procedure fs_utilities (var ecode : integer; var parms : ut_parmt);  (* replaces OSVM

procedure list_versions (var info : vers_info);

procedure lockseg (var errnum: integer);

procedure unlocksegs (var errnum: integer);

procedure unitio (var errnum : integer; devnum : integer; bufadr : longint;
                  numblocks : longint; blocknum : longint; op : ioop);
                     (* a subsitute routine for unitread and unitwrite *)

procedure monio (var ch : char; op : ioop);

procedure set_time (var ecode : integer;  time : time_rec);

procedure Change_Directory (var errnum : integer;  restartShell : boolean);

function LOGGING: boolean;

procedure LOG(var errnum: integer; ptr_arr: longint);

procedure LOG_NEWCMD(var errnum: integer; cmd: Tlog_cmds);

procedure Size_Stack(var errnum: integer; delta_size: longint);

procedure List_LibFiles (var errnum : integer;  var progfile : pathname;
                           var file_list : lf_list);
```

```
procedure LibFileName (var errnum : integer;  lf_num : integer;
                       var lf_name : pathname);

procedure Set_DebugMode (var errnum : integer;  on_off : boolean);

procedure Read_PMem (var errnum : integer;  var my_pMem : pMemRec );

procedure Write_PMem (var errnum : integer;  my_pMem : pMemRec);

procedure pset_file_info ( var ecode  : integer;
                               refnum : integer;
                               fsi    : fs_info );

procedure pset_vol_info ( var ecode  : integer;
                              path   : pathname;
                              fsi    : fs_info );

function OSBOOTVOL(var error : integer) : boot_tports;

procedure GET_CONFIG_NAME( var error    : integer;
                               devpostn : tports;
                           var devname  : e_name);

function DISK_LIKELY(var error    : integer;
                         devpostn : boot_tports) : boolean;

procedure CARDS_EQUIPPED(var error   : integer;
                         var in_slot : slot_array);
```

ADDITIONS TO THE OS

This section documents all the changes to the OS that have occurred since the last release of the OS Reference Manual. When the manual is updated, the material will be deleted from this section.

NEW OS PROCEDURES

A new data segment call named Info_Address was added in release 5.2. This call returns the refnum of the data segment that maps to the address that the user specifies in the call. The definition of the call is:

```
Info_Address (var errnum: integer;
                  address: longint;
              var refnum: integer);
```

where errnum is the return status of the call, address is the address that is converted to a data segment number, and refnum is the refnum of the data segment that maps to the address 'address'.

Info_Address can return any of the following errors:

303 - No data segment bound to the address

314 - Address does not fall within the valid ldsn range

1998 - Invalid parameter address

MEM_INFO, defined below retrieves information about the memory resources that the calling process uses.

```
MEM_INFO (var errnum : integer
          var swapspace;
              dataspace;
              cur_codesize;
              max_codesize: longint)
```

where:

swapspace = amount of system memory available (in bytes) for swapping

dataspace = amount of memory (in bytes) the calling process requires for its bound data areas. This value includes the stack of the process and the data segment for shared intrinsic data.

cur_codesize = size (in bytes) of the calling segment.

max_codesize = size (in bytes) of the longest code segment within the address space of the calling process.

The Set_File_Info system call, added in release 5.2, alters certain status information associated with a file system object. The call format is:

```
Set_File_Info (var ecode: integer;
                   refnum:integer;
                   fsi: fs_info);
```

where ecode is the error return code, refnum is the refnum of the object, and fsi is the fs information record that the OS calls LOOKUP and INFO use. Note that the object that refnum identifies must be open at the time of the call to Set_File_Info.

Set_File_Info alters the status information of the named object to conform to the values of the information record named in the call. Call INFO and set the values in fsi you want to change before calling this procedure.

Currently, Set_File_Info can change the following status fields:

- file_scavenged
- file_closed_by_OS
- file_left_open

DATA SEGMENT CHANGES

From OS release 5.1 on, OPEN_DATASEG is much less sensitive to the values of LEOF and PEOF within the data segment being opened. The results of an OPEN_DATASEG call under various conditions are outlined below:

Condition	Resulting Data Segment
0 < LEOF <= 128kb PEOF = any value	memory size = LEOF; disk size = PEOF errnum = 0
LEOF > 128kb PEOF = any value	errnum = 306 (data segment too big)
LEOF = 0 0 < PEOF <= 128kb	memory size = PEOF; disk size = PEOF errnum = -320 (a warning)
LEOF = 0 PEOF > 128kb	memory size = 128kb; disk size = PEOF errnum = -320 (a warning)
LEOF = 0 PEOF = 0	memory size = 512 ; disk size = 0 errnum = -320 (a warning)

Those conditons which result in a warning error (-320) should be checked via INFO_DATASEG to verify that the resulting data segment has the desired memory and disk sizes before the segment is used.

In release 5.2 of the Operating System the calling sequence for Make_Dataseg has changed to:

```
Tdstype = (ds_shared, ds_private)

Make_Dataseg (var errnum: integer;
              var segname: pathname;
                  mem_size, disc_size: longint;
              var refnum: integer;
              var segptr: integer;
                  ldsn :integer;
                  dstype : Tdstype);
```

All parameters are the same as before except dstype. dstype now specifies whether the data segment created is shared or private. The segment is located on the disk that segname specifies. Note that a null segname is no longer valid.

Info_Dataseg returns the attributes of a data segment in the record DsInfo. Release 5.2 adds two values to this record: segptr and volname. segptr is of type longint and represents the base address of the data segment; volname is of type e_name and represents the volume that contains the data segment.

CHANGES TO LISABUG

This section presents the additions and fixes to Lisabug that were made in the 5.2 release of the OS. These changes involve stack crawl and breakpoints.

The stack crawl command, SC, is now very robust. It correctly lists all stack frames even in domain 0.

This version of Lisabug allows you to run a program with the initial breakpoint set at the first instruction of the program. See the description of the D(ebug command in the section on the OS Shell in this manual for the details of this feature.

You can now set a breakpoint in code that is not in memory. As the code is swapped in and out, the OS automatically restores the breakpoints associated with the code currently in memory. Note that although you can set breakpoints in code that is not in memory, you can only use symbols for the code, such as procedure names, when the code is in memory. Therefore, you must use the logical address form of the breakpoint command when setting a breakpoint in code not in memory. However, once the code is swapped in, you can use the symbolic form of the breakpoint.

Another added breakpoint command feature is a format for specifying breakpoints on a process basis. The syntax of the new format is:

```
>br procid : address
```

When the process specified by procid reaches 'address', the breakpoint is taken. However, if any other process reaches 'address', the breakpoint is not taken. If you don't specify a procid in a breakpoint command, the OS assumes the procid of the currently executing process (most application's programmers will probably use this mode). Note that to reflect the change of associating a breakpoint with a process, Lisabug now displays the procid of the currently running process as well as the domain currrently in effect.

Although only the OS group will probably do this, if you wish to set a breakpoint in the Operating System, specify 0 as the procid in the command. Using 0 as the procid, causes all processes to take the breakpoint in the OS when they reach the specified address.

ERROR CHANGES AND ADDITIONS

Release 5.2 contains a new error for Open_Dataseg, Make_Dataseg, and Size_Dataseg. This error, 315, indicates that the operation could not be completed because it might cause a data lockout situation.

Open_Dataseg can now return the warning -321. This warning indicates that the data segment was opened successfully, but the file system returned a warning that the data within the segment may not be valid because the data segment was open when the system crashed.

OS ERROR MESSAGES

The following list of OS error messages is in ascending numerical order. However, the ordering scheme ignores the sign of the error number; the minus sign preceding an error number indicates that the message is a warning; the OS may or may not have completed the flagged action.

0	no error

PROCESS MANAGEMENT

100	Specified process does not exist
101	Specified process is a system process
110	Invalid priority specified (must be 1..255) (SetPriority_Process)
-115	Specified process is already suspended (Suspend_Process)
-120	Specified process is already active (Activate_Process)
-125	Sepcified process is already terminating (Kill_Process)
130	Could not open program file
131	Error while trying to read program file
132	Invalid program file (incorrect format)
133	Could not get a stack segment for new process
134	Could not get a syslocal segment for new process
135	Could not get a PCB for new process (no sysglobal space)
136	Could not set up communication channel for new process
138	Error accessing program file while loading
139	Could not get a PLCB to load the program (no sysglobal space)
141	Error accessing a library file while loading program (e.g. the library file containing required shared segment not found)
142	Can't run protected file on this machine
143	Program uses an intrinsic unit not found in the Intrinsic Library
144	Program uses an intrinsic unit whose name or type does not agree with the Intrinsic Library
145	Program uses a shared segment not found in the Intrinsic Library
146	Program uses a shared segment whose name does not agree with the Intrinsic Library
147	No space in syslocal for program file descriptor during process creation
148	No space in the shared IU data segment for shared IU globals required by the program

EXCEPTION MANAGEMENT

201	No such exception name declared
202	No space left in the system data area for declare_execp_hdl or signal_excep.
203	Null name specified as exception name.

MEMORY MANAGEMENT

302	Invalid ldsn
303	No data segment bound to an ldsn when there should be
304	Data segment bound to an ldsn when it shouldn't be
306	Data segment too large
307	Input data segment path name is invalid
308	Data segment already exists
309	Insufficient disk space for data segment
310	An invalid size has been specified: - memory size <= 0 - memory size of shared data segment > 128K - disk size < 0
311	Insufficient system resources
312	Unexpected file system error
313	Data segment not found
314	Invalid address passed to Info_Address
315	Operation may cause a data lockout
-320	Could not determine size of data segment. Defaults used were : memory size = 512 bytes, disk size = 0 bytes
-321	Data segment open when the system crashed. Data possibly invalid.

EVENT MANAGEMENT

401	Invalid event channel name passed to make_event_chn: empty string or string longer than 16 characters
402	No space left in system global data area for open_event_chn
403	No space left in system local data area for open_event_chn
404	Non-block structured device specified in pathname
405	Catalog is full in Make_Event_Chn or Open_Event_Chn
406	No such event channel exists in Kill_Event_Chn
410	Attempt to open a local event channel to send
411	Attempt to open an event channel to receive when event channel already has a receiver
-412	Event channel was left open and system crashed, Open_Event_Chn
-413	Event channel was scavenged, Open_Event_Chn
413	Unexpected file system error in Open_Event_Chn
416	Cannot get enough disk space for event channel in Open_Event_Chn
417	Unexpected file system error in Close_Event_Chn
420	Attempt to wait on a channel that the calling process did not open
421	Wait_Event_Chn returns while waiting on an empty channel because a sender process was not able to successfully complete sending an event.
422	Attempt to call wait_event_chn on an empty event-call channel
423	Cannot find corresponding event channel after being blocked (wait_event_chn)
424	The actual amount of data returned while reading an event from a channel is not the same as the size of that event

	block in wait_event_chn (probably disk I/O failure)
425	Event channel empty after being unblocked, Wait_Event_Chn
426	Bad request pointer error returned in Wait_Event_Chn
427	Wait_List has illegal length specified, Wait_Event_Chn
428	Receiver unblocked because last sender closed, Wait_Event_Chn
429	Unexpected file system error in Wait_Event_Chn
430	Attempt to send to a channel which the calling process does not have open
431	The actual amount of data transferred while writing an event to a channel is not the same as the size of an event block in send_event_chn (disk is probably full)
432	Sender unblocked because receiver closed in Send_Event_Chn
433	Unexpected file system error in Send_Event_Chn
440	Unexpected file system error in Make_Event_Chn
441	Event channel already exists in Make_Event_Chn
445	Unexpected file system error in Kill_Event_Chn
450	Unexpected file system error in Flush_Event_Chn
600	See GENERAL I/O ERRORS below

TWIGGY DISK

606	Can't find sector (disk unformatted)
608	See GENERAL I/O ERRORS below
609	See GENERAL I/O ERRORS below
610	See GENERAL I/O ERRORS below
614	No disk present in drive
617	Checksum error
618	Can't format, or write-protected

GENERAL I/O

600	Attempt to perform I/O operation on non I/O request
605	Call to non-configured device driver
608	Illegal length or disk address for transfer
609	Call to non-configured device driver
610	No more room in Sysglobal for I/O request
615	Wrong call version to Twiggy
616	Unpermitted Twiggy function
619	No more room in Sysglobal for I/O request
658	Premature end of file when reading from driver
659	Corrupt file system header chain found in driver

TIME MANAGEMENT

630	The time passed to delay_time, convert_time, or send_event_chn is such that the year is less than 1900 or greater than 2035.
631	Illegal Timeout request parameter
634	Illegal Timed event id of -1
635	Process got unblocked prematurely due to process termination (delay_time)
636	Timer request did not complete successfully (delay_time)
638	The time passed to delay_time or send_event_chn is more than 23 days from the current GMT time

639	Illegal date passed to Set_Time, or illegal date from system clock in Get_Time

RS-232

640	RS-232 driver called with wrong version number
641	RS-232 read or write initiated with illegal parameter
642	Unimplemented or unsupported RS-232 driver function
643	Unexpected RS-232 interrupt
646	No memory available to initialize RS-232
647	Unexpected RS-232 timer interrupt
648	Attempt to send unpermitted command to serial controller card
649	Illegal device control parameters to RS-232

PROFILE DISK

652	N-port driver not initialized prior to Profile
653	No room in sysglobal to initialize Profile
654	Hard error status returned from drive
655	Wrong call version to Profile
656	Unpermitted Profile function
658	See GENERAL I/O ERRORS above
659	See GENERAL I/O ERRORS above
660	Cable disconnected
662	Parity error
663	Checksum error
666	Timeout
670	Bad command response from drive
685	Eject not allowed this device

SERIAL DRIVER

680	Wrong call version to serial driver
682	Unpermitted serial driver function
683	No room in sysglobal to initialize serial driver

N-PORT CARD DRIVER

686	No room in sysglobal to initialize n-port card driver
687	Unpermitted n-port card driver function
688	Wrong call version to n-port card driver

PARALLEL PRINTER

690	Wrong call version to parallel printer
691	Illegal parallel printer parameters
692	N-port card not initialized prior to parallel printer
693	No room in sysglobal to initialize parallel printer
694	Unimplemented device control
696	Out of paper
698	Offline

STARTUP

700	Mismatch between loader version number (in OS.OBJ) and operating system version number (in SYSTEM.OS.OBJ)
701	OS exhausted its internal space during startup
702	Cannot make system process
703	Cannot kill pseudo-outer process
704	Cannot create driver
705	Cannot program NMI key
706	Cannot (soft) initialize Twiggy
707	Cannot (soft) initialize the file system volume
708	Profile not readable
709	Cannot map screen data
710	Too many slot-based devices

FILE SYSTEM

VmStuff:

801	IoResult <> 0 on I/O using the Monitor (LISAIO)
802	Asynchronous I/O request not completed successfully
803	Bad combination of mode parameters (this is an internal error that should not occur when you run your code)
806	Page specified is out of range (TFDM)
809	Invalid arguments (page, address, offset, or count) (VM)
810	The requested page could not be read in (VM)
816	Not enough sysglobal space for file system buffers (initqvm)
819	Bad device number (IO_INIT)
820	No space in sysglobal for asynchronous request list
821	Already initialized I/O for this device
822	Bad device number (IO_DISINIT)

SFileIO:

825	Error in parameter values (Allocate)
826	No more room to allocate pages on device
828	Error in parameter values (Deallocate)
829	Partial deallocation only (ran into unallocated region)
835	s-file number < 0 or > maxfiles (illegal value) (SList_IO)
837	Unallocated s-file or I/O error (FMap_Mgr)
838	Map overflow: s-file too large (this error obsolete from release 5.2 of the OS on)
839	Attempt to compact file past PEOF (FMap_Mgr)
841	Unallocated s-file or I/O error (Get_PSize)
843	Requested exact fit, but one couldn't be provided (AppendPages)
847	Requested transfer count is <= 0 (DataIO)
848	End-of-file encountered
849	Invalid page or offset value in parameter list
852	Bad unit number (FlushFS)
854	No free slots in s-list directory (too many s-files) (New_SFile)
855	No available disk space for file hints
856	Device not mounted
857	Empty, locked, or invalid s-file (Kill_SFile)
861	Relative page is beyond PEOF (bad parameter value) (AbsPage)
864	No sysglobal space for volume bitmap (Real_Mount, Real_Unmount)
866	Wrong FS version or not a valid Lisa FS volume
867	Bad unit number (Real_Mount, Real_Unmount)

```
   868          Bad unit number (Def_Mount, Def_Unmount)
   869          Unit already mounted (mount)/no unit mounted (unmount)
   870          No sysglobal space for DCB or MDDF (mount)
 FS_Primitives:
   871          Parameter not a valid s-file ID (Open_SFile)
   872          No sysglobal space for s-file control block
   873          Specified file is already open for private access
   874          Device not mounted
   875          Invalid s-file ID or s-file control block (Close_SFile)
   879          Attempt to postion past LEOF (Direct_IO)
   881          Attempt to read empty file (FileIO)
   882          No space on volume for new data page of file
   883          Attempt to read past LEOF
   884          Not first auto-allocation, but file was empty
   885          Could not update filesize hints after a write (fileio)
   886          No syslocal space for I/O request list
   887          Catalog pointer does not indicate a catalog (bad parameter)
   888          Entry not found in catalog (Lookup_by_ename)
   890          Entry by that name already exists (Make_Entry)
   891          Catalog is full, or was not as catalog
   892          Illegal name for an entry
   894          Entry not found, or not a catalog (Kill_Entry)
   895          Invalid entry name (kill_entry)
   896          Safety switch is on--cannot kill entry (kill_entry)
 FS_Init:
   897          Invalid bootdev value
 FS_Interface:
   921          Pathname invalid or no such device (Make_File)
   922          Invalid label size (Make_File)
   926          Pathname invalid or no such device (Make_Pipe)
   927          Invalid label size (Make_Pipe)
   941          Pathname invalid or no such device (Kill_Object)
   946          Pathname invalid or no such device (Open)
   947          Not enough space in syslocal for file system refdb
   948          Entry not found in specified catalog (Open)
   949          Private access not allowed if file already open shared
   950          Pipe already in use, requested access not possible OR
                dwrite not allowed for pipe
   951          File is already opened in private mode (open)
   952          Bad refnum (Close_Object)
   954          Bad refnum (Read_data)
   955          Read access not allowed to specified object
   956          Attempt to position FMARK past LEOF not allowed
   957          Negative request count is illegal (read_data)
   958          Non-sequential access is not allowed (read_data)
   959          System resources exhausted
   960          Error writing to pipe while an unsatisfied read was pending
   961          Bad refnum (write_data)
   962          No WRITE or APPEND access allowed
   963          Attempt to position FMARK too far past LEOF
   964          Append access not allowed in absolute mode
   965          Append access not allowed in relative mode
   966          Internal inconsistency of FMARK and LEOF (warning)
   967          Non-sequential access is not allowed (write_data)
```

968	Bad refnum (Flush)
971	Pathname invalid or no such device (Lookup)
972	Entry not found in specified catalog
974	Bad refnum (Info)
977	Bad refnum (allocate)
978	Page count is non-positive (allocate)
979	Not a block structured device (allocate)
981	Bad refnum (Truncate)
982	No space has been allocated for specified file
983	Not a block structured device (truncate)
985	Bad refnum (Compact)
986	No space has been allocated for specified file
987	Not a block structured device (compact)
988	Bad refnum (Flush_Pipe)
989	Caller is not a reader of the pipe
990	Not a block structured device (flush_pipe)
994	Invalid refnum (Set_File_Info)
995	Not a block-structured device (Set_File_Info)
999	Asynchronous read was unblocked before it was satisfied. This may occur during process termination.
1021	Pathname invalid or no such entry (Rename_Entry)
1022	No such entry found (rename_entry)
1023	Invalid newname, check for '>' in string (rename_entry)
1024	New name already exists in catalog (rename_entry)
1031	Pathname invalid or no such entry (Read_Label)
1032	Invalid transfer count (read_label)
1033	No such entry found (read_label)
1041	Pathname invalid or no such entry (Write_Label)
1042	Invalid transfer count (write_label)
1043	No such entry found (write_label)
1051	No device or volume by that name (mount)
1052	A volume is already mounted on device
1053	Attempt to mount the temporarily unmounted boot volume just unmounted from this machine (MOUNT)
-1063	Warning, attempt to mount a temporarily unmounted boot volume that was either unmounted from another machine or was not the most recently unmounted boot volume. The mount is completed (MOUNT)
1061	No device or volume by that name (Unmount)
1062	No volume is mounted on device
1071	Not a valid or mounted volume for working directory
1091	Pathname invalid or no such entry (Set_Safety)
1092	No such entry found (set_safety)
1121	Invalid device, not mounted, or not a catalog (reset_catalog)
1128	Invalid pathname, device, or volume not mounted (get_dev_name)
1130	File is protected; cannot open due to protection violation
get_open_list	
1131	No device or volume by that name
1132	No volume is mounted on that device
1133	No more open files in the file list of that device (no files, data segments, event channels open on that device)
reg_open_list	
1134	Cannot find space in sysglobal for open file list
1135	Cannot find the open file entry to modify

fs_utilities calls:

1136	Boot volume not mounted (fs_utility, ubd)
1137	Boot volume already unmounted (fs_utility, ubd)
1138	Caller cannot have higher priority than system processes when calling ubd (fs_utility, ubd)
1141	Boot volume was not unmounted when calling rbd
1142	Some other volume still mounted on the boot device when calling rbd
1143	No sysglobal space for MDDF to do rbd
1144	Attempt to remount a volume which is not the temporarily unmounted boot volume from the same machine (rbd)
1145	No sysglobal space for bit map to do rbd
1159	fs_shutdown is not allowed while boot volume unmounted but operation is carried out

fs_shutdown calls:

1158	Track-by-track copy buffer is too small
1159	Shutdown requested while boot volume was unmounted
1160	Destination device too small for track-by-track copy
1161	Invalid final shutdown mode
1162	Power is already off

fs_utilities calls:

1163	Illegal command
1164	Device is not a Twiggy device
1165	No volume is mounted on the device
1166	A valid volume is already mounted on the device
1167	The Device is not blockstructured
1168	Device name is invalid

newvolume (volume initialization):

1169	Could not default mount volume before initialization
1170	Could not mount volume after initialization
1171	'-' is not allowed in a volume name
1172	No space available to initialize a bitmap for the volume

WARNINGS! from opening a file or mounting a volume:

-1173	File was last closed by the OS
-1174	File was left open or volume was left mounted, and system crashed
-1175	File or volume was scavenged

When these warnings occur on an OPEN call for a file or a MOUNT call for a volume, the OS goes ahead and opens the volume/file for access as usual. HOWEVER, the contents of the file might be inconsistent.

CIRCULAR PIPES:

1176	Cannot read from a pipe more than half of the allocated physical size (read_data)
1177	Cannot cancel a read request for a pipe (read_data)
1178	Process waiting in read_data for pipe data got unblocked because the last writer of the pipe has closed it (read_data)
1180	Cannot write to a pipe more than half of the allocated physical size (write_data)
1181	No system space left for request block for pipe (write_data)

1182	Writer process to a pipe got unblocked before the request was satisfied (this can occur during process termination) (write_data)
1183	Cannot cancel a write request for a pipe (write_data)
1184	Process waiting in write_data for pipe space got unblocked because the reader closed the pipe (write_data)
1186	Cannot allocate space to a pipe while it has data wrapped around (allocate)
1188	Cannot compact a pipe while it has data wrapped around (compact)
1190	Attempt to access a page that is not allocated to the pipe (absrelbyte)

OTHER:

1196	Something is still open on device--cannot unmount (real_unmount)
1197	Volume is not formatted or cannot be read (def_mount)
1198	Negative request count is illegal (write_data)
1199	Function or procedure is not yet implemented
1200	Illegal volume parameter
1201	Blank file parameter
1202	Error writing destination file
1203	Invalid UCSD directory
1204	File not found
1210	Boot track program not executable
1211	Boot track program too big
1212	Error reading boot track program
1213	Error writing boot track program
1214	Source file not found
1215	Can't write boot tracks on that device
1216	Couldn't create/close internal buffer
1217	Boot track program has too many code segments
1218	Couldn't find configuration information entry
1219	Couldn't get enough working space
1220	Premature EOF in boot track program
1221	Position out of range
1222	No device at that position
1998	Invalid parameter address
1999	Bad refnum

The pathname error codes (921, 926, 941, 946, and 971) often mean that the volume specified in the pathname is not mounted. If error 966 occurs while writing a file using the FTP utility, you probably ran out of space on the destination volume.

Operating System Error Codes by Procedure

PROCESS MANAGEMENT

Note that Yield_CPU and Terminate_Process return no errors

Returned by all procedures except Make_Process

100	Specified process does not exist
101	Specified process is a system process

SetPriority_Process

110	Invalid priority specified (must be 1..255)

Suspend_Process

-115	Specified process is already suspended

Activate_Process

-120	Specified process is already active

Kill_Process

-125	Specified process is already terminating

Make_Process

130	Could not open program file
131	Error while trying to read program file
132	Invalid program file (incorrect format)
133	Could not get a stack segment for new process
134	Could not get a syslocal segment for new process
135	Could not get a PCB for new process (no sysglobal space)
136	Could not set up communication channel for new process
138	Error accessing program file while loading
139	Could not get a PLCB to load the program (no sysglobal space)
141	Error accessing a library file while loading program (e.g. library file containing shared segment required by program not found)
142	Can't run protected file on this machine
143	Program uses an intrinsic unit not found in the Intrinsic Library
144	Program uses an intrinsic unit whose name or type does not agree with the Intrinsic Library
145	Program uses a shared segment not found in the Intrinsic Library
146	Program uses a shared segment whose name does not agree with the Intrinsic Library
147	No space in syslocal for program file descriptor during process creation
148	No space in the shared IU data segment for shared IU globals required by the program

EXCEPTION MANAGEMENT

Returned by all procedures
1998 Invalid parameter address

Declare_excep_hdl
201 No such exception name declared
202 No space left in the system data area
203 Null name specified as exception name.

Disable_excep
201 No such exception name declared
203 Null name specified as exception name.

Enable_excep
201 No such exception name declared
203 Null name specified as exception name.

Info_excep
201 No such exception name declared
203 Null name specified as exception name.

Flush_excep
201 No such exception name declared
203 Null name specified as exception name.

Signal_excep
201 No such exception name declared
202 No space left in the system data area
203 Null name specified as exception name.

MEMORY MANAGEMENT
Returned by all procedures
1998 Invalid parameter address

Returned by all procedures except INFO_LDSN, MAKE_DATASEG, OPEN_DATASEG, KILL_DATASEG, and MEM_INFO
1999 Bad refnum

Note that SETACCESS_DATASEG and INFO_DATASEG return only 1998 and 1999 and that MEM_INFO returns only 1998

INFO_LDSN
302 Invalid ldsn
303 No data segment bound to an ldsn when there should be

UNBIND_DATASEG
303 No data segment bound to an ldsn when there should be

BIND_DATASEG
302 Invalid ldsn
304 Data segment bound to an ldsn when it shouldn't be

```
MAKE_DATASEG
  302       Invalid ldsn
  304       Data segment bound to an ldsn when it shouldn't be
  306       Data segment too large
  307       Input data segment path name is invalid
  308       Data segment already exists
  309       Insufficient disk space for data segment
  310       An invalid size has been specified:
               - memory size <= 0
               - disk size < 0
  311       Insufficient system resources
  312       Unexpected file system error
  315       Possible data lockout

OPEN_DATASEG
  302       Invalid ldsn
  304       Data segment bound to an ldsn when it shouldn't be
  306       Data segment too large
  307       Input data segment path name is invalid
  311       Insufficient system resources
  312       Unexpected file system error
  313       Data segment not found
  315       Possible data lockout
 -320       Warning: could not determine size of data segment.
            The following defaults were used:
               - memory size = 512 bytes
               - disk size   =   0 bytes
 -321       Data segment open when system crashed

CLOSE_DATASEG
  312       Unexpected file system error

KILL_DATASEG
  307       Input data segment path name is invalid
  312       Unexpected file system error
  313       Data segment not found

SIZE_DATASEG
  304       Data segment being grown into an LDSN already in use
  306       Data segment too large
  309       Insufficient disk space for data segment
  310       An invalid size has been specified:
               - memory size <= 0
               - memory size of shared data segment > 128K
               - disk size < 0
  312       Unexpected file system error
  315       Possible data lockout

FLUSH_DATASEG
  312       Unexpected file system error

INFO_ADDRESS
  303       No data segment bound to the address
  314       Address does not fall within valid ldsn range
```

EVENT MANAGEMENT

Returned by all procedures

1998	Invalid parameter address

Make_Event_Chn

401	Invalid event channel name passed to Make_Event_Chn: empty string or string longer than 16 characters
404	Non-block structured device specified in pathname to Make_Event_Chn, Kill_Event_Chn, or Open_Event_Chn
405	Catalog is full in Make_Event_Chn
440	Unexpected file system error in Make_Event_Chn
441	Event channel already exists in Make_Event_Chn

Kill_Event_Chn

401	Invalid event channel name passed to Kill_Event_Chn: empty string or string too long
404	Non-block structured device specified in pathname
406	No such event channel exists in Kill_Event_Chn
445	Unexpected file system error in Kill_Event_Chn

Open_Event_Chn

201	No such exception name declared
401	Invalid event channel name passed to Open_Event_Chn: empty string or string too long
402	No space left in system global data area for Open_Event_Chn
403	No space left in system local data area for Open_Event_Chn
404	Non-block structured device specified in pathname
406	No such event channel exists in Open_Event_Chn
411	Attempt to open an event channel to receive when event channel already has a receiver
-412	Event channel was left open and system crashed, Open_Event_Chn
-413	Event channel was scavenged, Open_Event_Chn
413	Unexpected file system error in Open_Event_Chn
416	Cannot get enough disk space for event channel at open

Returned when the event channel is local:

405	Catalog is full in Open_Event_Chn
410	Attempt to open a local event channel to send

Close_Event_Chn

201	No such exception name declared
417	Unexpected file system error in Close_Event_Chn
1999	Bad refnum

Info_Event_Chn

1999	Bad refnum

Wait_Event_Chn

402	No space left in system global data area
420	Attempt to wait on a channel that the calling process did not open
421	Wait_Event_Chn returns while waiting on an empty channel because a sender process was not able to successfully complete sending an event
422	Attempt to call Wait_Event_Chn on an empty event-call channel
423	Cannot find corresponding event channel after being blocked
424	The actual amount of data returned while reading an event from a channel is not the same as the size of an event block in Wait_Event_Chn (probably disk I/O failure)
425	Event channel empty after being unblocked
426	Bad request pointer error return from Can_Aread_Pipe
427	Wait_List has illegal length specified, Wait_Event_Chn
428	Receiver unblocked because last sender closed, Wait_Event_Chn
429	Unexpected file system error in Wait_Event_Chn
1999	Bad refnum

Flush_Event_Chn

450	Unexpected file system error in Flush_Event_Chn
1999	Bad refnum

Send_Event_Chn

430	Attempt to send to a channel which the calling process does not have open
431	The actual amount of data transferred while writing an event to a channel is not the same as the size of an event block in Send_Event_Chn (disk is probably full)
432	Sender unblocked because receiver closed in Send_Event_Chn
433	Unexpected file system error in Send_Event_Chn
1999	Bad refnum

TIME MANAGEMENT

Returned by all procedures:
(Note that this is the only error message that Set_Local_Time_Diff returns)

1998	Invalid parameter address

Delay_Time

630	The time passed to Delay_Time, Convert_Time, or Send_Event_Chn is such that the year is less than 1900 or greater than 2035
632	No space in sysglobal
635	Process got unblocked prematurely due to process termination (Delay_Time)
636	Timer request did not complete successfully
638	The time passed to Delay_Time or Send_Event_Chn is more than 23 days from the current GMT time

Convert_Time

630	The time passed to Delay_Time, Convert_Time, or Send_Event_Chn is such that the year is less than 1900 or greater than 2035

Get_Time

639	Year not between 1981 and 1995 in Get_Time or Set_time. In Get_Time the error indicates a dead battery.

Set_Time

639	Year not between 1981 and 1995 in Get_Time or Set_Time.

PWBT

1210	Boot track program not executable
1211	Boot track program too big
1212	Error reading boot track program
1213	Error writing boot track program
1214	Source file not found
1215	Can't write boot tracks on that device
1216	Couldn't create/close internal buffer
1217	Boot track program has too many code segments
1218	Couldn't find configuration information entry
1219	Couldn't get enough working space
1220	Premature EOF in boot track program

PFTP

1200	Illegal volume parameter
1201	Blank file parameter
1202	Error writing destination file
1203	Invalid UCSD directory
1204	File not found

Get_Config_Name

1221	Position out of range
1222	No device at that position

Disk_Likely

1221	Position out of range
1222	No device at that position

OS LOADER DIAGNOSTICS

Error Message	Cause or Description
FILE SYSTEM VERSION MISMATCH	The boot tracks don't know the right file system version
FILE SYSTEM CORRUPT	Either damaged file system or damaged contents
MEMORY EXHAUST	The OS will not fit
SYSTEM CODE FILE NOT FOUND	Cannot find SYSTEM.OS
SYSTEM CONFIGURATION FILE NOT FOUND	Cannot find SYSTEM.CONFIG
BOOT DEVICE READ FAILED	Device could not be read for whatever reason
CODE FILE CORRUPT	Refers to SYSTEM.OS
TOO MANY OS SEGMENTS	Refers to SYSTEM.OS
SYSTEM DEBUG FILE NOT FOUND	Cannot find SYSTEM.DEBUG
PROGRAM NOT EXECUTABLE	Refers to SYSTEM.OS, SYSTEM.DEBUG or SYSTEM.LLD
SYSTEM LOW LEVEL DRIVER FILE NOT FOUND	Refers to SYSTEM.LLD
CONFIGURATION FILE NOT USABLE	Refers to SYSTEM.CONFIG
WRONG DRIVER	For instance, storing a Twiggy driver on a Profile
RANGE ERROR, OR UNKNOWN BOOT ERROR	A loader bug

SYSTEM ERRORS

A system error indicates that something has gone seriously awry within the Operating System code. When a system error occurs, the Operating System reports the error and stops. Please report the occurrence of any system errors to the Operating System group.

Common system errors:

10102	Error while creating System.Shell during StartUp
10201	Hardware exception (divide by zero, for example) in Operating System code

EXCEPTIONS

During execution applications can field hardware exceptions. If such an exception occurs, the system displays one of the following messages:

Bus error or address error exception:

```
EXCEPTION in process of gid <gggg>
Process is about to be terminated.
access address = <aaaaaaaa> = mmu# <mmm> (segment name), offset <oooo>
inst reg = <rrrr>       sr = <ssss>     pc = <pppppp>
saved registers at <xxxxxxxx>
Going to Lisabug, type g to continue
```

Any other hardware exception:

```
EXCEPTION in process of gid <gggg>
Process is about to be terminated.
sr = <ssss>     pc = <pppppp>
saved registers at <xxxxxxxx>
Going to Lisabug, type g to continue
```

where:

<gggg> is the global ID of the process that incurred the exception.
<aaaaaaaa> is the address that caused the bus or address error
<mmm> is the segment number represented by <aaaaaaaa> and
<oooo> is the offset within that segment
<rrrr> is the value of the instruction register at the time of the exception
<ssss> is the value of the status register at the time of the exception
<pppppp> is the value of the program counter at the time of the exception
<xxxxxxxx> is the address of the saved register information

All numbers displayed are decimal; the segment name is displayed only if the segment number makes sense to the Operating System.

If the exception is divide by zero, overflow, or CHK out of bounds, the process is not terminated and the line to that effect is not shown. If the process has declared an exception handler for this exception, that handler is entered after you type g to LisaBug, and the process then continues execution. If no handler has been declared, the system default handler terminates the process. If the exception is a bus error and the segment name is 'stack seg', a stack overflow has probably occurred. The Operating System cannot currently recover from this error.

If the exception occurs in Operating System code, the displays are the same as given above except that the first two lines are replaced by:

EXCEPTION in system code!

If you type g in Lisabug after this exception, a system error 10201 occurs and you must reboot.

You should use release 7.4 or later of the Monitor because in these versions the Lisabug register display is the user's register display and the user can use the stack crawl command to find the calling procedures. You should not examine the memory location <xxxxxx> that contains the saved registers because the debugger saves the system's registers there.