EDUCATIONAL
SERVICES

VAX/VMS
Training

VAX/VMS
Device Driver
Optional
Driver Routines

digital

EY-2278E-MI-0001

OPTIONAL DRIVER ROUTINES

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, October 1984

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

FIGURES

EXAMPLES

INTRODUCTION

The routines listed below need not be present in a driver. Their presence is determined by the physical characteristics of the device (when certain device registers need to be initialized). These routines ease the coding of other routines (making the UCB for a single-unit controller the owner UCB in the IDB, thereby eliminating the need for REQPCHAN and RELCHAN macros), or supply information (producing error log reports about the device). These routines include:

- Timeout Routine
- Cancel I/O Routine
- Controller Initialization Routine
- Unit Initialization Routine
- Register Dump Routine (General Error Logging)

OBJECTIVES

Upon completion of this module, you will be able to:

1. Write the following driver routines:
   - timeout
   - cancel I/O
   - unit initialization
   - controller initialization
   - register dump
2. Incorporate error logging capabilities in a driver.
3. Interpret the error reports generated by ANALYZE/ERROR_LOG which relate to your driver.

RESOURCES

1. Guide to Writing a Device Driver for VAX/VMS

2. VAX/VMS Guide to System Management and Daily Operations

LEARNING ACTIVITIES

1. Study drivers which have unit initialization routines, controller initialization routines, cancel I/O routines, and register dump routines (i.e., those that perform error-logging<.

2. Create an error log report.

TOPICS

- Timeout routine
- Powerfail recovery routine
- Cancel I/O routine
- Controller initialization routine
- Unit initialization routine
- Error logging

TIMEOUT ROUTINE

The driver timeout routine is invoked:

- by the system timer routine, running at IPL = 8 (was invoked at IPL = 7).
- when the timeout interval, specified in a WFIKPCH or WFIRLCH macro, expires.
- after a powerfail

The system timer routine runs once every second. It scans all UCBs that have timeouts enabled (UCB$V_TIM), to see if a timeout should occur, and updates all due time fields for UCBs expecting a timeout. Before calling the driver timer routine, the system timer routine:

- clears bits UCB$V_INT and UCB$V_TIM in UCB$W_STS to disable interrupts or another timeout.
- sets the timeout bit UCB$V_TIMOUT in UCB$W_STS.
- sets IPL to device IPL.
- restores the saved fork process context.
- calls the timeout routine (via JSB).

The driver timeout routine normally does one of the following:

- retries the request (a fixed number of times).
- aborts the I/O request.
- sends a message to an operator, and waits.

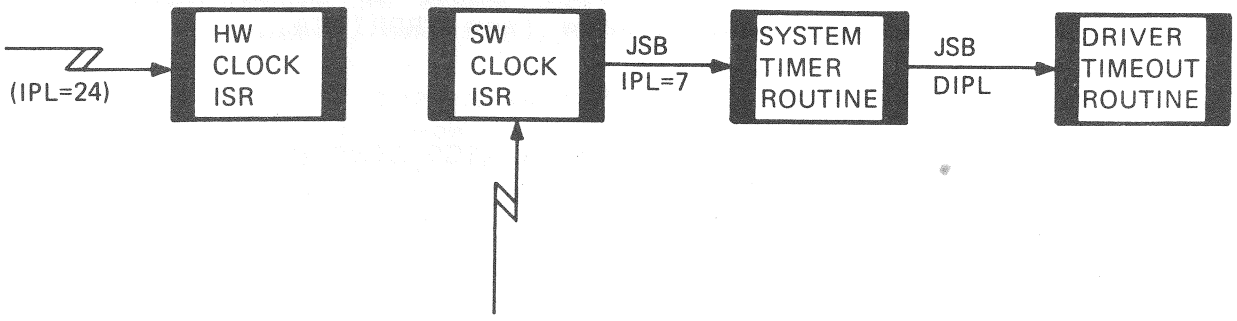

Figure 9-1 Timer and Timeout Sequence

A timeout routine may perform some of the following:

- Modifies device status register(s) (e.g., clears error bits).

- Sets IPL to fork level with SETIPL UCB$B_FIPL(R5), not using IOFORK. A synchronization problem could arise if IOFORK is used, since the fork dispatcher, at fork IPL, when returning control would try to return control to the timer routine (at IPL = 8). IPL cannot be lowered in this fashion. By setting IPL to fork level (with SETIPL) the fork dispatcher is never invoked, since it is invoked only by IOFORK.

- Returns system resources (e.g., data path, mapping registers).

- Checks for the cancel bit (UCB$V_CANCEL in UCB$W_STS). If set,

  - places SS$_ABORT or SS$_CANCEL in RØ.
  - writes other status informaiton in RØ and R1.
  - invokes REQCOM.

- Checks the powerfail bit (UCB$V_POWER in UCB$W_STS). If set, attempts powerfail recovery, if possible (discussed later).

- Logs a device timeout error by calling ERL$DEVICTMO (error logging is discussed in another module).

- Updates a retry counter (stored in the UCB). If threshold is exhausted, proceeds as for cancel, otherwise, clears UCB$V_TIMOUT in UCB$W_STS, and retries the operation.

- Sends a message to an operator and retries the operation (perhaps sending the message after a fixed number of timeouts). To send a message (see CRDRIVER):

  - save R3 and R4 on stack.

  - load a message code such as MSG$_DEVOFFLIN into R4.

  - load the address of the operator's mailbox into R3 (e.g., SYS$GL_OPRMBX).

  - send the message using JSB G^EXE$SENDEVMSG

  - restore R3 and R4 from stack.

POWERFAIL RECOVERY

After a powerfailure, routine EXE$RESTART in module POWERFAIL of the executive performs the following:

- raises IPL to 31.
- sets bit UCB$V_POWER in UCB$W_STS in all UCBs.
- calls all controller initialization routines.
- calls all unit initialization routines.
- for each UCB with either UCB$V_INT or UCB$V_TIM set (in UCB$W_STS):
  - clears UCB$V_INT
  - sets UCB$V_TIM
  - clears UCB$L_DUETIM

In effect, the unit will timeout, and the driver timeout routine is entered.

When a powerfailure is detected by the timeout routine, the routine can either abort the operation (if no recovery is possible), or retry the entire operation (if that is possible for the particular function code for the device). When retrying the operation, all counters, flags, and relevant fields in the UCB should be restored to their original state (possibly by recopying them from the IRP). Also, the address of the IRP may have been destroyed, so as a precaution, the driver should restore the IRP address

```
MOVL  UCB$L_IRP(R5), R3
```

If a powerfailure is detected before the WFIKPCH macro is issued, but after a DSBINT macro, then an ENBINT macro should be issued to remove the saved IPL from the stack. Device registers should be reloaded, and the I/O operation retried.

In any case, the driver is responsible for clearing the UCB$V_POWER bit in UCB$W_STS. Sometimes, drivers clear this bit when starting an I/O operation, in case a powerfail occurred while the device was inactive (yet the powerfail bit was still set).

CANCEL I/O ROUTINE

The address of the cancel I/O routine is placed in the DDT. The cancel I/O routine is called whenever

- a channel to the device is deassigned.
- the device is deallocated.
- a $CANCEL system service is issued. (forced by VMS on image exit and process deletion)

When called, the cancel I/O routine may expect

- to be at fork IPL
- R2 = negated value of channel index number
- R3 = IRP address
- R4 = PCB address for process issuing $CANCEL
- R5 = UCB address
- R8 = Reason for call (i.e., CANC_CANCEL,CANC_DASSGN; see $CANDEF

Before being called, all IRPs queued to the UCB from the specific process and with the specified channel number are removed from the queue, and have SS$_CANCEL placed into IRP$L_MEDIA before they are placed on the I/O postprocessing queue. Therefore, the cancel I/O routine need only concern itself with the current I/O request.

Normally, DMA devices do not have a cancel I/O entry point since I/O operations cannot be stopped once a transfer has started.

Frequently, cancel I/O routines simply contain a

```
JMP G^IOC$CANCELIO
```

which verifies that

- the unit is busy.
- the current IRP is for the process requesting $CANCEL.
- the current IRP is for the channel over which the $CANCEL was issued.

If the above conditions are met, IOC$CANCELIO sets the UCB$V_CANCEL bit in UCB$W_STS. The timeout and/or start I/O routines should check for this bit being set, and terminate the I/O operation after putting appropriate status information into R0 and R1.

Another common use of the cancel I/O routine is to restore the UCB fields to their original states when a user is done with the device (resetting flags, counters, etc.), turning the unit on/off-line, clearing error bits, etc. To tell when a user is deassigning/deallocating the device, test for the reference count (going to Ø for nonshareable devices, or decreasing by 1 for shareable devices). For example:

```
TSTW  UCB$W_REFC(R5)          ; nonshareable device
BEQL  GOING_AWAY              ; being deassigned/
                              ; deallocated
```

Note that R4 does not contain the CSR address, and if your cancel routine will request some device activity (and call on WFIKPCH), you will need to save and restore the PCB address passed in R4 (probably in an extension to the UCB).

CONTROLLER INITIALIZATION ROUTINE

The Controller Initialization Routine is invoked

- at IPL$_POWER (31).
- when unit CONNECTed or driver RELOADed (SYSGEN).
- after powerfailure.
- R4 = CSR address.
- R5 = IDB address.
- R6 = DDB address.
- R8 = CRB address.

The processing involved is device-dependent, but may include:

- initializing fields in CRB or IDB (e.g., VEC addresses in CRB that are more than 256 bytes away from start of CRB, and therefore cannot be initialized by DPT_STORE macro).
- initialize owner field (IDB$L_OWNER) of the IDB so REQPCHAN and RELCHAN macros are not necessary (for single-unit controllers).

- permanently allocating resources such as data paths or mapping registers (e.g., setting bit VEC$V_PATHLOCK in CRB$L_INTD + VEC$B_DATAPATH to permanently allocate a data path, after calling REQDPR, or VEC$V_MAPLOCK in CRB$L_INTD + VEC$W_MAPREG to permanently allocate a mapping register, after calling REQMPR). Be careful to check for powerfailure first, so resources are not doubly allocated (UCB$V_POWER in UCB$W_STS).

UNIT INITIALIZATION ROUTINE

The Unit Initialization Routine is invoked

- at IPL$_POWER (31).
- when unit is CONNECTed (SYSGEN).
- after powerfailure.
- R3 = CSR address (primary).
- R4 = CSR address (secondary, differs for MASSBUS devices only, when it contains the MBA CSR address).
- R5 = UCB address.

The processing involved is also device-dependent. The same things can be done in the unit initialization routines as in the controller initialization routines. Usually, the unit initialization routine sets bits in the UCB and the owner field of the IDB, while the controller initialization routines initialize fields in the CRB and IDB. Among the operations not listed in the controller initialization section are:

- clear error bits in device register(s).
- setting device on-line (UCB$V_ONLINE in UCB$W_STS).

To perform unit initialization, and to initialize fields used in the start I/O routine, a MASSBUS driver must locate the device registers for the unit. For a single-device MASSBUS controller, the following steps are taken:

- Extract the unit number from UCB$W_UNIT and store it in UCB$B_SLAVE. For example:

```
MOVZWL UCB$W_UNIT(R5), R3       ; get drive unit number
MOVB R3, UCB$B_SLAVE(R5)        ; set slave unit number
```

- Multiply the slave number by 32 to derive the longword offset to the device registers for the driver (store the result in UCB$B_SLAVE+1). For example:

```
MULL #<1@7>/4, R3
MOVB R3, UCB$B_SLAVE+1(R5)
```

- Assuming that the offset to the device registers is in R3, and R4 contains the MBA CSR, the driver can load the address of the device registers into a general register as follows:

```
MOVAL MBA$L_ERB(R4)[R3],R3
```

(MBA$L_ERB is a fixed offset to the start of the external registers.) The configuration register is the first register in the space reserved for MBA registers.

For a multidevice MASSBUS controller, the driver uses R3, R4, and R5 (passed by the driver loading procedure) to locate the device registers:

- Computes the MBA unit number of the controller by using R3 to determine the number of bytes from the start of MBA address space to the device registers, and dividing the result by 128. The result is stored in UCB$B_SLAVE.

  For example:

```
SUBL3 R4,R3,R2                  ; calculate offset to
                                ; drive control register
MOVAB -MBA$L_ERB(R2),R2         ; subtract out external
                                ; register base
DIVW3 #1@7,R2,UCB$B_SLAVE(R5)   ; set adapter driver
                                ; number
```

- Stores the drive offset constant in UCB$B_SLAVE+1 (equals slave value multiplied by 32). For example:

  ```
  MULB3 #1@7/4,UCB$B_SLAVE(R5), UCB$B_SLAVE+1(R5)
  ```

- Performs any initialization functions required.

DRIVER ERROR LOGGING

- Is an optional driver feature.
- Is a cooperative effort between a driver, system subroutines, an error formatting process (ERRFMT), and a report generation program invoked with ANALYZE/ERROR_LOG.
- Provides a mechanism whereby device errors and device timeouts can be recorded, and later analyzed.

Error Logging Components

Figure 9-2 summarizes the flow of data and control among the major components involved in error logging. The following notes are keyed to the figure.

Device Error:

1. Driver detects a device error and calls system routine ERL$DEVICERR.
2. ERL$DEVICERR
   - logs device error.
   - allocates error message buffer space for error log buffer.
   - fills in pertinent information (device class and type, etc.).
   - calls driver register dump routine.
3. Driver Register Dump Routine
   - dumps device register contents in allocated error message buffer.
   - returns to ERL$DEVICERR.
4. ERL$DEVICERR returns control to driver.

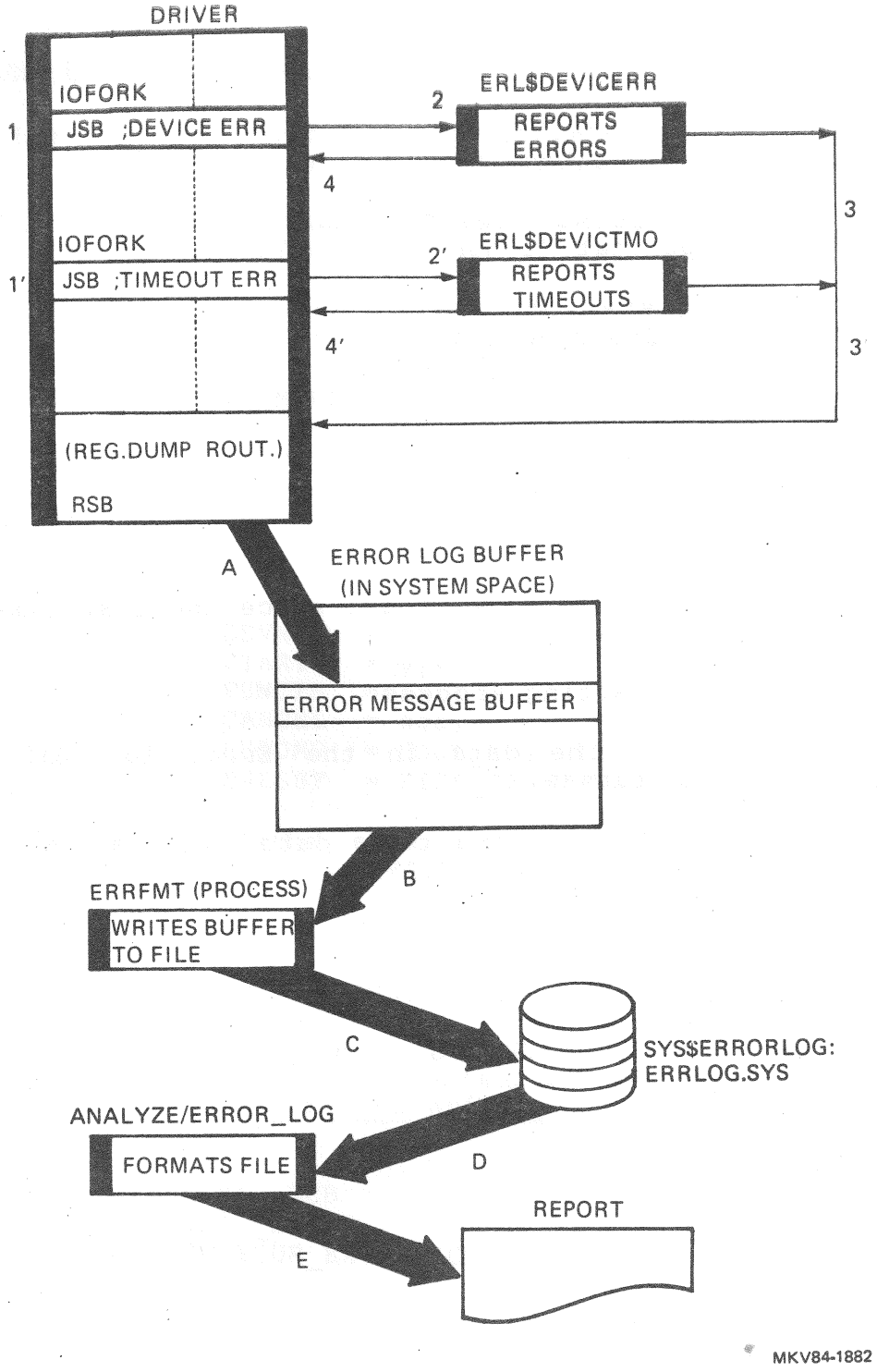

Figure 9-2 Control and Data Flow of Driver Error Logging

Timeout Error:

1' Driver detects a device timeout error and calls a system routine ERL$DEVICTMO.

2',3',4' ERL$DEVICTMO does the same thing as ERL$DEVICERR except that it marks the error as a timeout error.

A. Driver Register Dump routine dumps device register contents in Error Message Buffer.

B. The Error Formatting Process ERRFMT is awakened:

- if Error Log Buffer is full, or
- if ten messages have been written into the Error Log Buffer, or
- if 30 seconds have elapsed since the last time ERRFMT was awakened.

C. The Error Formatting Process

- transforms the data in the Error Log Buffer into standard format.
- writes the formatted data on a disk file SYS$ERRORLOG: ERRLOG.SYS.

D,E. The User

- renames ERRLOG.SYS to ERRLOG.OLD.
- use ANALYZE/ERROR_LOG command to generate a report from ERRLOG.OLD

Necessary Steps for Error Logging

1. Include the $EMBDEF macro to define EMB symbols for your driver (not in template driver).

2. In the driver prologue table, set the DEV$V_ELG bit in the UCB$L_DEVCHAR field:

```
        DPT_STORE   INIT
        DPT_STORE   UCB, UCB$L_DEVCHAR, L, <-
                    DEV$M_AVL!-          ; device available
                    DEV$M_ELG>           ; enable error log.
        DPT_STORE   REINIT
        DPT_STORE   END
```

3. In the driver dispatch table, specify a register dump routine and an error log buffer size:

```
        DDTAB   -
                DEVNAM   = xxx, -
                START    = yyy, -
                FUNCTB   = zzz, -
                CANCEL   = aaa, -
                REGDMP   = YOUR_REGISTER_DUMP_ROUTINE_ADDR,-
                ERLGBF   = SIZE_OF_ERROR_LOG_BUFFER
```

The size of the error log buffer is computed in the following way:

```
SIZE_OF_ERROR_LOG_BUFFER = 4 (for register count) +
                           4  *  (number  of  registers
                           saved) + EMB$L_DV_REGSAV (for
                           system overhead)
```

4. Include the error log extension to the UCB (see Appendix A of the Guide to Writing a Device Driver for VAX/VMS Manual):

```
        $DEFINI  UCB
        .=UCB$L_DPC+4              ; include error log ext.
$DEF    UCB$W_YOUR_EXTENSIONS .BLKW 1
            .                      ; to hold register dump
            .                      ; data
            .
$DEF    UCB$K_MY_UCBLEN            ; length of extended UCB
        $DEFEND  UCB
```

[In the driver prologue table, specify
UCBSIZE = <UCB$K_MY_UCBLEN>]

In your driver, place driver register contents, or what is written to driver registers, in the extensions reserved for each register (before an I/O operation is initiated, or after the I/O operaton completes).

5. Within your driver, code JSB calls to system routines for reporting device errors or device timeouts. These routines should be called at fork IPL.

```
    JSB  G^ERL$DEVICERR        ; for device errors

or  JSB  G^ERL$DEVICTMO        ; for device timeouts
```

6. Write a register dump routine which writes the number of device registers to be dumped, and their current contents, into the error buffer. This routine is called by ERL$DEVICERR or ERL$DEVICTMO.

   When called, the routine may assume that:

   R0 = address of buffer into which device registers are dumped.

   R4 = address of the controller status register (CSR), assuming that the driver used the WFIKPCH macro to wait for an interrupt or timeout.

   R5 = address of the device's UCB. (Store the registers to be dumped in user-defined extensions to the UCB.)

The routine should save and restore R3-R11 if it requires their use.

The routine begins by writing the number of device registers to be dumped into the buffer (as a longword value).

Then all the device registers are placed into the buffer (each one as a longword value).

The register dump routine exits with an RSB to return control to ERL$DEVICERR or ERL$DEVICTMO.

Example 9-1 is a register dump routine that saves four register values. The register values were placed into fields in the UCB before calling ERL$DEVICERR or ERL$DEVICTMO. (Note that the fields are user-defined extensions to the UCB.) The symbol PT_NUM_REGS has been equated to 4 earlier in the driver.

```
11620           .SBTTL  PT_REG_DUMP, Device register dump routine
11640
11650   ;++
11660   ; PT_REG_DUMP, Dumps the contents of device registers to a buffer
11670   ;
11680   ; Functional description:
11690   ;
11700   ;       Writes the number of device registers, and their current
11710   ;       contents into a diagnostic or error buffer.  Reader and
11720   ;       punch both have two registers (a status register, and a
11730   ;       data register).
11740   ;
11750   ; Inputs:
11760   ;
11770   ;       R0      - address of the output buffer
11780   ;       R4      - address of the CSR (controller status register)
11790   ;       R5      - address of the UCB (unit control block)
11800   ;
11810   ; Outputs:
11820   ;
11830   ;       The routine must preserve all registers except R1-R3.
11840   ;
11850   ;       The output buffer contains the current contents of the device
11860   ;       registers. R0 contains the address of the next empty longword in
11870   ;       the output buffer.
11880   ;
11890   ;--
11900
11910   PT_REG_DUMP:                                    ; Dump device registers
11920           MOVZBL  #PT_NUM_REGS,(R0)+              ; STORE DEVICE REGISTER COUNT
11930           MOVZWL  UCB$W_PT_PPS(R5),(R0)+          ; STORE PUNCH STATUS REGISTER
11940           MOVZBL  UCB$B_PT_PPB(R5),(R0)+          ; STORE PUNCH DATA REGISTER
11943           MOVZWL  UCB$W_PT_PRS(R5),(R0)+          ; STORE READER STATUS REGISTER
11946           MOVZBL  UCB$B_PT_PRB(R5),(R0)+          ; STORE READER DATA REGISTER
11950           RSB                                     ; Return
```

Example 9-1 Register Dump Routine

Notes on Error Logging

- Enabling/disabling error logging may be implemented by a driver. Usually, an IO$_SETCHAR function is provided which sets/clears the DEV$V_ELG bit in UCB$L_DEVCHAR field.

- Error logging may be enabled on a per $QIO basis. This is typically performed:

  - at the beginning of the start I/O routine. Move the function code specified by the user (IRP$W_FUNC) into UCB$W_FUNC (part of the UCB error log extension)

  - if the user has issued a $QIO function with the function modifier IO$M_INHERLOG, error logging is inhibited for that $QIO.

- When the driver calls the system routines ERL$DEVICERR or ERL$DEVICTMO, an error logging buffer is allocated, and its address is stored in UCB$L_EMB.

- Device class and device type (UCB$B_DEVCLASS and UCB$B_DEVTYPE) are written into the error logging buffer by the system routines.

- The bit UCB$V_ERLOGIP (in UCB$W_STS) is set by the system routines to indicate error logging is in progress. The bit is cleared by IOC$REQCOM after storing the device status, error count, and I/O status block in the error logging buffer.

Obtaining Error Reports

The following steps may be taken to obtain an error report. For full details, see the VAX/VMS System Management and Operations Guide.

1. Set the default to SYS$ERRORLOG

2. Rename ERRLOG.SYS to ERRLOG.OLD.

3. Use ANALYZE/ERROR_LOG to generate a report.

The following commands were used to obtain the SYE report in Example 9-2.

```
__   INPUT FILE?              file spec for ERRLOG.OLD

__   OUTPUT FILE?             any file spec (later PRINTed)

__   OPTIONS?      S          (standard)

__   DEVICE NAME? UNKNOWN     [all UNKNOWN device entries]

__   AFTER DATE?   __ <CR>    [first entry]

__   BEFORE DATE?  __ <CR>    [last entry]
```

The driver had an error log routine that dumped two registers. The error log routines were called with the first register to be dumped containing 200 (hex), and the second register containing 20 (hex). Three errors were logged.

The following notes are keyed to Example 9-2:

1. If ERL$DEVICTMO is called, the entry is identified as a device timeout, rather than a device error. Otherwise, the entry contains the same information.

2. Fields of interest in the IRP and UCB (at the time the error logging routines are called) are displayed.

3. Note that 21 (hex) corresponds to the IO$_READLBLK function code.

4. The device name is always stored in a 16-byte field.

5. These are the registers dumped by the driver routine. First is the count (2), followed by the first register, then the second register.

```
V A X / V M S          SYSTEM ERROR REPORT        COMPILED 26-JUN-1980 14:23
                                                                PAGE   1.

****************************** ENTRY     2. ******************************
UNKNOWN DEVICE TYPE, DEVICE ERROR            LOGGED 19-JUN-1980 14:34:32.08
ERROR SEQUENCE 1738.                         SYSTEM ID REGISTER     118700D

ERROR LOG RECORD

      ERF$L_SID          0118700D

      ERL$W_ENTRY            0001

      EXE$GQ_SYSTIME     AE87C100
                         00885138
      ERL$GL_SEQUENCE        06CA

      UCB$B_ERTCNT             00

      UCB$B_ERTMAX             00

      IRP$Q_IOSB         0000008C
                         00000000
      UCB$W_STS              0110

      UCB$B_DEVCLASS           64

      UCB$B_DEVTYPE            64

      IRP$L_PID          0024000D

      IRP$W_BOFF             0030

      IRP$W_BCNT             0024

      UCB$L_MEDIA        00000000

      UCB$W_UNIT             0000

      UCB$W_ERRCNT           0001

      UCB$L_OPCNT        00000000

      UCB$L_OWNUIC       00000000

      UCB$L_DEVCHAR      04440001

      UCB$B_SLAVE              00

      IRP$W_FUNC             0021

      DDB$T_NAME         41435003
                         00000000
                         00000000
                         00000000

      LONGWORD 1.        00000002

      LONGWORD 2.        00000200

SYSTEM ID REGISTER

ERROR ENTRY TYPE

64 BIT TIME WHEN ERROR LOGGED

UNIQUE ERROR SEQUENCE = 1738.

REMAINING RETRIES = 0.

MAXIMUM RETRIES = 0.

FINAL IOSB

FINAL DEVICE STATUS

DEVICE CLASS = 100.

DEVICE TYPE = 100.

REQUESTING PROCESS ID

TRANSFER BYTE OFFSET = 48.

TRANSFER BYTE COUNT = 36.

DEVICE DEPENDANT PHYSICAL ADDRESS

PHYSICAL UNIT NUMBER = 0.

UNIT ERROR COUNT = 1.

UNIT OPERATION COUNT = 0.

VOLUME OWNER UIC = [000,000]

DEVICE CHARACTERISTICS

DEVICE SLAVE CONTROLLER = 0.

QIO FUNCTION CODE

/.PCA...../

Device Registers
```


Example 9-2 Sample SYE Error Report

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED 26-JUN-1980 14:25
                                                          PAGE   2.

      LONGWORD 3.     00000020 }   (5)

****************************** ENTRY     3. ******************************
UNKNOWN DEVICE TYPE, DEVICE ERROR          LOGGED 19-JUN-1980 14:34:38.71
ERROR SEQUENCE 1739.                       SYSTEM ID REGISTER     118700D

ERROR LOG RECORD

      ERF$L_SID         0118700D
                                    SYSTEM ID REGISTER
      ERL$W_ENTRY           0001
                                    ERROR ENTRY TYPE
      EXE$GQ_SYSTIME    B27B6960
                        00885138    64 BIT TIME WHEN ERROR LOGGED
      ERL$GL_SEQUENCE       06CB
                                    UNIQUE ERROR SEQUENCE = 1739.
      UCB$B_ERTCNT            00
                                    REMAINING RETRIES = 0.
      UCB$B_ERTMAX            00
                                    MAXIMUM RETRIES = 0.
      IRP$Q_IOSB        0000008C
                        00000000    FINAL IOSB
      UCB$W_STS             0110
                                    FINAL DEVICE STATUS
      UCB$B_DEVCLASS          64
                                    DEVICE CLASS = 100.
      UCB$B_DEVTYPE           64
                                    DEVICE TYPE = 100.
      IRP$L_PID         0024000D
                                    REQUESTING PROCESS ID
      IRP$W_BOFF            0030
                                    TRANSFER BYTE OFFSET = 48.
      IRP$W_BCNT            0024
                                    TRANSFER BYTE COUNT = 36.
      UCB$L_MEDIA       00000000
                                    DEVICE DEPENDANT PHYSICAL ADDRESS
      UCB$W_UNIT            0000
                                    PHYSICAL UNIT NUMBER = 0.
      UCB$W_ERRCNT          0002
                                    UNIT ERROR COUNT = 2.
      UCB$L_OPCNT       00000001
                                    UNIT OPERATION COUNT = 1.
      UCB$L_OWNUIC      00000000
                                    VOLUME OWNER UIC = [000,000]
      UCB$L_DEVCHAR     04440001
                                    DEVICE CHARACTERISTICS
      UCB$B_SLAVE             00
                                    DEVICE SLAVE CONTROLLER = 0.
      IRP$W_FUNC            0021
                                    QIO FUNCTION CODE
      DDB$T_NAME        41435003
                        00000000
                        00000000
                        00000000
                                    /.PCA...../
      LONGWORD 1.       00000002
```

Example 9-2 Sample SYE Error Report (Cont)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED 26-JUN-1980 14:26
                                                          PAGE   3.

      LONGWORD 2.     00000200

      LONGWORD 3.     00000020

****************************** ENTRY     4. ******************************
UNKNOWN DEVICE TYPE, DEVICE ERROR            LOGGED 19-JUN-1980 14:34:42.57
ERROR SEQUENCE 1740.                         SYSTEM ID REGISTER     118700D

ERROR LOG RECORD

      ERF$L_SID        0118700D
                                    SYSTEM ID REGISTER
      ERL$W_ENTRY          0001
                                    ERROR ENTRY TYPE
      EXE$GQ_SYSTIME   B4C866A0
                       00885138     64 BIT TIME WHEN ERROR LOGGED
      ERL$GL_SEQUENCE      06CC
                                    UNIQUE ERROR SEQUENCE = 1740.
      UCB$B_ERTCNT           00
                                    REMAINING RETRIES = 0.
      UCB$B_ERTMAX           00
                                    MAXIMUM RETRIES = 0.
      IRP$Q_IOSB       0000008C
                       00000000     FINAL IOSB
      UCB$W_STS            0110
                                    FINAL DEVICE STATUS
      UCB$B_DEVCLASS         64
                                    DEVICE CLASS = 100.
      UCB$B_DEVTYPE          64
                                    DEVICE TYPE = 100.
      IRP$L_PID        0024000D
                                    REQUESTING PROCESS ID
      IRP$W_BOFF           0030
                                    TRANSFER BYTE OFFSET = 48.
      IRP$W_BCNT           0024
                                    TRANSFER BYTE COUNT = 36.
      UCB$L_MEDIA      00000000
                                    DEVICE DEPENDANT PHYSICAL ADDRESS
      UCB$W_UNIT           0000
                                    PHYSICAL UNIT NUMBER = 0.
      UCB$W_ERRCNT         0003
                                    UNIT ERROR COUNT = 3.
      UCB$L_OPCNT      00000002
                                    UNIT OPERATION COUNT = 2.
      UCB$L_OWNUIC     00000000
                                    VOLUME OWNER UIC = [000,000]
      UCB$L_DEVCHAR    04440001
                                    DEVICE CHARACTERISTICS
      UCB$B_SLAVE            00
                                    DEVICE SLAVE CONTROLLER = 0.
      IRP$W_FUNC           0021
                                    QIO FUNCTION CODE
      DDB$T_NAME       41435003
                       00000000
                       00000000
                       00000000
                                    /.PCA............./
```

Example 9-2 Sample SYE Error Report (Cont)

```
V A X / V M S        SYSTEM ERROR REPORT      COMPILED 26-JUN-1980 14:28
                                                          PAGE    4.

      LONGWORD 1.     00000002

      LONGWORD 2.     00000200

      LONGWORD 3.     00000020


SUMMARY TOTALS

      UBA INTERRUPTS                          0.
      HARD ECC ERRORS                         0.
      ASYNCHRONOUS WRITE ERRORS               0.
      SOFT ECC ERRORS                         0.
      SBI ALERT ERRORS                        0.
      SBI FAULT ERRORS                        0.
      MACHINE CHECKS                          0.
      DEVICE ERRORS                           0.
      SYSTEM INFORMATION                      1.
      CONFIGURATION CHANGES                   0.
      VECTOR AREA PROBLEMS                    0.
      SOFTWARE ERRORS                         0.

      RECORDS PROCESSED                       3.
      RECORDS READ                            4.

      UNKNOWN/ILLEGAL ENTRIES                 3.

      DATE OF EARLIEST ENTRY          19-JUN-1980 14:33:14.84
      DATE OF LATEST ENTRY            19-JUN-1980 14:33:14.84

PROGRAM RUNTIME STATISTICS

      TIMES IN SECONDS        PAGE     DIRECT     BUFFERED
          CPU  ELAPSED       FAULTS       I/O          I/O

          1.8    345.8          380         1          253

%SYE-I-SUCCOM, SUCCESSFUL COMPLETION
$
```

Example 9-2 Sample SYE Error Report (Cont)

digital

EY-2278E-MI-0001
Printed in U.S.A.