

```

EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AAAAAA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSSS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MMMM MMMM PP PP LL EE SS
EE XX XX AA AA MM MM MM PP PP LL EE SS
EE XX XX AA AA MM MM MM PP PP LL EE SS
EE XX XX AA AA MM MM MM PP PP LL EE SS
EEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSS
EEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSS
EEEEEEEE XX XX AA AA MM MM PPPPPPPP LL EEEEEEEEE SSSSSSS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AAAAAAAAAA MM MM PP LL EE SS
EE XX XX AA AA MM MM PP LL EE SS
EE XX XX AA AA MM MM PP LL EE SS
EE XX XX AA AA MM MM PP LL EE SS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS
EEEEEEEEEE XX XX AA AA MM MM PP LLLLLLLLLL EEEEEEEEE SSSSSSSS

```

```

LL      PPPPPPP  MM      MM  UU      UU  LL      TTTTTTTTTT
LL      PPPPPPP  MM      MM  UU      UU  LL      TTTTTTTTTT
LL      PP      PP  MMMM  MMMM  UU      UU  LL      TT
LL      PP      PP  MMMM  MMMM  UU      UU  LL      TT
LL      PP      PP  MM  MM  MM  UU      UU  LL      TT
LL      PP      PP  MM  MM  MM  UU      UU  LL      TT
LL      PPPPPPP  MM      MM  UU      UU  LL      TT
LL      PPPPPPP  MM      MM  UU      UU  LL      TT
LL      PP      MM      MM  UU      UU  LL      TT
LL      PP      MM      MM  UU      UU  LL      TT
LL      PP      MM      MM  UU      UU  LL      TT
LL      PP      MM      MM  UU      UU  LL      TT
LLLLLLLLLLL  PP      MM      MM  UUUUUUUUUU  LLLLLLLLLLL  TT
LLLLLLLLLLL  PP      MM      MM  UUUUUUUUUU  LLLLLLLLLLL  TT

```

```

88888888  333333  222222
88888888  333333  222222
88      88  33      33  22      22
88      88  33      33  22      22
88      88  33      33  22      22
88      88  33      33  22      22
88888888  33      22
88888888  33      22
88      88  33      22
88      88  33      22
88      88  33      22
88      88  33      22
88888888  333333  2222222222
88888888  333333  2222222222

```

MODULE lpmultast(%TITLE'line printer driver' MAIN=lp\_main, IDENT='V04-000')=

```

*****
*
* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
* ALL RIGHTS RESERVED.
*
* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
* TRANSFERRED.
*
* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*
*****

```

++

FACILITY:

A sample program that illustrates the use of the connect to interrupt facility.

ABSTRACT:

This program assigns a channel to a line printer device, and then connects to the device via the connect to interrupt facility. The program then requests the name of a file from the user, and outputs that file on the line printer.

AUTHOR:

Alan Lehotsky 12-Oct-1979

REVISION HISTORY:

Revised for publication	APL	21-Mar-1980
-------------------------	-----	-------------

--

LPM  
%SB  
ROU  
++  
L  
F  
  
I  
  
O  
  
--  
  
END

```
%SBTTL 'External and local symbol definitions'
BEGIN
```

```
LIBRARY 'SYSSLIBRARY:LIB';           ! Get important definitions
```

```
PSECT
```

```
! Define some PSECTS which we will need to refer to later
```

```
OWN=      sharedata(ALIGN(9),WRITE),
OWN=      data;
```

```
LINKAGE
```

```
intrupt=  JSB(REGISTER=2, REGISTER=4, REGISTER=5):
           NOPRESERVE(0,1,2,3,4) NOTUSED(6,7,8,9,10,11),
cancel=    JSB(REGISTER=2, REGISTER=3, REGISTER=4, REGISTER=5):
           NOTUSED(6,7,8,9,10,11);
```

```
FORWARD ROUTINE
```

```
lp_interrupt: intrupt PSECT(sharedata),           ! Interrupt server
lp_cancel:    NOVALUE cancel PSECT(sharedata),    ! Cancel I/O
lp_main,
lp_isr_ast,
lp_iodone_ast;
```

```
! Static Definitions
```

```
LITERAL
```

```
true = 1,
false = 0,
```

```
io_page_count = 1,           ! Pages needed in UNIBUS I/O space
```

```
io_space_base = %x'20100000', ! Physical address of UBA 0 space
! for VAX-11/780. Other processors
! need different magic number...
```

```
unibus_lp_addr = %0'777514', ! 18-bit addr of LP11 CSR
```

```
! Calculate the page-frame number to map to get the physical address
! that the unibus is mapped on.
```

```
io_page_pfn = (io_space_base + unibus_lp_addr)/512,
```

```
lp_csr_offset = unibus_lp_addr and 511, ! Offset to printer CSRs.
```

```
filename_length = 100,
```

```
record_bufsiz = 256,
prompt_length = 28;
```

```

OWN
lpchan:      WORD,                . Line printer channel number.

filename_buffer: VECTOR[filename_length, BYTE],
file_descr:  VECTOR[2] INITIAL( filename_length, filename_buffer),
fdlen:      WORD,

record_buffer: VECTOR[record_bufsiz, BYTE],

file_fab:    $fab(                ! Input file fab
             fac=get,
             fna=filename_buffer,
             org=seq,
             rfm=var,
             dnm='TEST.LIS'),

file_rab:    $rab(
             fab=file_fab,
             rac=seq,
             ubf=record_buffer,
             usz=record_bufsiz),

io_page_limits: VECTOR[2]        : Addresses of process-mapped
             INITIAL(200,        : UNIBUS I/O page. 200 tells $CRMPSC
             200);              : to map pages in PO space

```

```

BIND
onesecond_delta=                : Delta time format for one
  UPLIT(-10*1000*1000,-1);      : second.

```

```

: Define offsets into the buffer that will be shared by the user
: process and the process routines that execute in kernel mode.
:

```

```

FIELD
buf=
  SET
  buf$l_flags= [0,0,32,0],       ! Flags longword.
  buf$v_int=   [0,0,1,0],       ! Interrupt expected

  buf$w_charcount=[4,0,16,0],   ! Number of chars in buffer
  buf$l_startdata=[8,0,32,0]   ! Start of data in buffer.
  TES,

lp=
  SET
  lp_csr=      [0,0,16,1],      ! Offset to line printer CSR
  lp_dbr=      [2,0,8,0]       ! Offset to line printer data
  TES;

```



```
%SBTTL 'LP_INTERRUPT, Interrupt service routine'
```

```
ROUTINE lp_interrupt( arglist, idb, ucb ): intrupt PSECT(sharedata)=
```

```
++
Functional description:
```

This routine services an interrupt from the line printer device. If the interrupt was expected, the routine disables output interrupts. The disable is an optimization to prevent one interrupt per character. With output interrupts disabled, the line printer buffers characters until the device needs to output the characters. Then the main program enables output interrupts only for the period of time necessary for the device to empty the buffer.

Then the interrupt service routine loads a success status into R0 and returns.

If the interrupt was not expected, the routine just loads an error status into R0 to prevent delivery of an AST to the owning process and returns.

```
Inputs:
```

```
R2      - address of a counted argument list
R4      - address of the IDB
R5      - address of the UCB
```

```
The counted argument list is as follows:
```

```
0(R2)  - count of arguments (4)
4(R2)  - the system-mapped address of the user buffer
8(R2)  - the system-mapped address of the device's CSR
12(R2) - the IDB address
16(R2) - the UCB address
```

```
Outputs:
```

```
The routine must preserve all registers except R0-R4.
```

```
--
BEGIN
MAP
  arglist:  REF VECTOR[ ,LONG],
  ucb:      REF BLOCK[ ,BYTE],
  idb:      REF BLOCK[ ,BYTE];
BIND
  bufadr = arglist[1]:  REF BLOCK FIELD(buf);  ! System adr of buffer
BUILTIN
  TESTBITCC;
IF TESTBITCC( bufadr[buf$l_flags] )
THEN
  RETURN 0;  ! No interrupt expected, no AST wanted
(.idb[idb$l_csr])<0,16> = 0;  ! Disable the output interrupt
```





```
%SBTTL 'LP_CANCEL, Cancel I/O on Line Printer'
```

```
ROUTINE lp_cancel( chan_idx, irp, pcb, ucb ): NOVALUE cancel PSECT(sharedata)=
```

```
  * functional description:
```

```
    This routine disables output interrupts from the line printer.
```

```
Inputs:
```

```
  R2 - negated value of the channel index number
  R3 - address of the current IRP (I/O request packet)
  R4 - address of the PCB (process control block) or the
      process canceling I/O
  R5 - address of the UCB (unit control block)
```

```
Outputs:
```

```
  none
```

```
--
BEGIN
MAP
```

```
  irp: REF BLOCK[.BYTE],
  pcb: REF BLOCK[.BYTE],
  ucb: REF BLOCK[.BYTE];
```

```
BIND
```

```
  crb= .ucb[ucb$l_crb]: BLOCK[.BYTE];
```

```
LOCAL
```

```
  csr: REF BLOCK[.BYTE] FIELD(lp); ! UNIBUS addr.
```

```
  csr = ..(crb[crb$l_intd] + BLOCK[0, vec$l_idb;0,BYTE]); ! Addr of CSR
```

```
  csr[lp_csr] = 0 ! Disable output interrupts.
```

```
END;
```

```
ADDI
```

```
: L
```

```
: Q
```

```
: :
```

```
: P1=
```

```
: P2=
```

```
: P3=
```

```
: P4=
```

```
: P5=
```

```
: P6=
```

```
: DI
```

```
: :
```

```
: DAC
```

```
: MAX
```

```
: MAX
```

```
: MAX
```

```
: ADC
```

```
: DI
```

```
: :
```

```
$DEI
```

```
$DEI
```

```
=:
```

```
$DE
```

```
: :
```

```
: DI
```

```
: :
```

%SBTTL \_P\_MAIN, the main routine'

ROUTINE lp\_main: PSECT(\$CODE\$)=

LP\_MAIN, the routine that controls the others

Functional description:

1. Assign a channel to the 'line printer.
2. Map the process to the I/O page.
3. Issue a connect to interrupt QIO to get the line printer.
4. Prompt the user for a file name.
5. Open and connect to the file.
6. Write the contents of the file to the line printer.

Inputs:

none

Outputs:

R0 - status code  
       SS\$NORMAL - success  
       RMS\_code - error in opening or reading  
                   the file  
       SS\$DEVOFFLINE - error is writing to printer

```

--
BEGIN
PSECT
OWN= $OWNS;
OWN
buffer_desc: VECTOR[2] INITIAL(
                    512+512,
                    output_buffer),
                    ! Descriptor of buffer shared
                    ! by process and kernel mode
                    ! process routines.

entry_list: VECTOR[4] INITIAL(
                    C,
                    0,
                    lp_interrupt-output_buffer,
                    lp_cancel-output_buffer);
                    ! List of offsets to kernel
                    ! mode routines: init device;
                    ! start device;
                    ! interrupt servicing;
                    ! cancel I/O.

LOCAL
csr: REF BLOCK[,BYTE] FIELD(lp) VOLATILE,
status;

EXTERNAL ROUTINE
lib$get_input;

```

Assign a channel to the line printer.

```

status = $assign(
    devnam=$DESCRIPTOR('LPA0'),
    chan=lpchan);
    ! Assign channel to line
    ! printer

```

ADD

.=U

\$DE

\$DE

\$DE

\$DE

UCB

: A

IRP

IRP

```
IF NOT .status THEN RETURN .status;
```

```
Map the UNIBUS I/O page to the process so that the line printer's
device registers are accessible.
```

```
status = Scmpsc(          ! Map I/O page to process.
  inadr=io_page_limits,
  retadr=io_page_limits,
  flags=sec$m_wrt OR sec$m_pfnmap OR sec$m_expreg,
  pagcnt=io_page_count,
  vbn=io_page_pfn );
```

```
IF NOT .status THEN RETURN .status;
```

```
Issue a connect to interrupt QIO to the line printer device. This
connection will allow the program to control and handle interrupts
from the device.
```

```
status = $qio(          ! Connect the process to the
  chan=.lpchan,         ! line printer device.
  func=io$_conintread, ! Specify a read only buffer.
  astadr=lp_iodone_ast, ! Specify an AST routine.
  p1=buffer_desc,      ! Specify a shared buffer.
  p2=entry_list,       ! Specify routine entry points.
  p3=cin$m_isr OR cin$m_cancel,
                        ! Specify ISR, cancel routines.
  p4=lp_isr_ast,       ! Specify an interrupt AST.
  p6=5);               ! Specify an AST count.
```

```
IF NOT .status THEN RETURN .status;
```

```
Ask user what file to print.
```

```
status = lib$get_input( file_descr,
  $descriptor('Name of file to be printed: '),
  file_descr[0]);
```

```
IF NOT .status THEN RETURN .status;
```

```
Open and connect file.
```

```
file_fab[fab$b_fns] = .file_descr[0];      ! Length of spec.
status = $open(fab=file_fab);              ! Open file.
IF NOT .status THEN RETURN .status;
```

LPMULT.B32;1

16-SEP-1984 16:59:26.90 Page 10

```
status = Sconnect( rab = file_rab );  
IF NOT .status THEN RETURN .status;
```

! Connect file.

ADD

: AI  
:

AD\_I

Get a record at a time until end of file. Surround record's contents with a linefeed and a carriage return.

```

WHILE status = $get(rab=file_rab) DO
  BEGIN
  LOCAL
    inp,
    outp;

  outp = output_buffer[buf$l_startdata]; ! Target for first character
  CH$WCHAR_A( %CHAR(%X'A'), outp); ! Start with a line-feed

  inp = record_buffer;

  ! Load length of this output buffer in the buffer header. Then copy
  ! the contents of the input buffer to the output buffer. Translate all
  ! lower case alphabetic to upper case characters.

  output_buffer[buf$w_charcount] = .file_rab[rab$w_rsz] + 2;
  DECR i FROM .file_rab[rab$w_rsz]-1 TO 0 DO
    BEGIN
    LOCAL
      char;

    char = CH$RCHAR_A( inp );
    SELECTIONE .char OF
      SET
      [%C'a' TO %C'z']: char = .char - %X'20'; ! Uppcase
      TES;

    CH$WCHAR_A( .char, outp )
    END;

  CH$WCHAR_A( %CHAR(%X'0D'), outp ); ! Put CR at end.

```

Send characters one at a time to the line printer. Before sending a character, see if the line printer is still in ready state. If not, set a timer to go off in one second, and go to sleep. When an AST occurs -- either because of a line printer interrupt, or because the timer runs out, the AST routine will wake the process up again.

If the line printer is still in ready state, just send the next character.

```

outp = output_buffer[buf$l_startdata]; ! Addr of output string
csr = .io_page_limits + lp_csr_offset; ! Addr of LP's CSR

```

```

DECR i FROM .output_buffer[buf$w_charcount]-1 TO 0 DO
  WHILE 1 DO
    BEGIN
      BIND
        devbits= csr[lp_csr]: VOLATILE SIGNED WORD;

      CASE SIGN(.devbits) FROM -1 TO 1 OF
        SET
        [-1]: RETURN ss$_devoffline;           ! Paper problem, maybe
        [1]: BEGIN                               ! Output a character
              csr[lp_dbr] = CHRCHAR_A( outp );
              EXITLOOP                          ! Back for next char
              END;
        [0]:
          !+
          ! Line printer is not ready. See whether it's in
          ! trouble, or just busy. If it's in trouble, stop
          ! program with error status. Otherwise, just wait
          ! until it comes ready again.
          !-
          BEGIN
            output_buffer[buf$v_int] = true;           ! Interrupt expected
            csr[lp_csr] = .csr[lp_csr] OR %X'40';      ! Enable LP interrupts
            status = $setimr(                          ! Set a one second timer.
              daytim=onesecond_delta,
              astadr=lp_isr_asf);
          END

          IF NOT .status THEN RETURN .status;

          $hiber;                                     ! Go to sleep.
          $cantim()                                  ! Cancel timer request
          END
        TES
      END

    END;                                     ! End $GET loop

  IF .status NEQ ss$_endoffile
  THEN
    RETURN .status;

  $close( fab=file_fab )

  END.

```

```

: C
: T
:
10$
20$
: C
:
30$

```

%%BTTL 'LP\_ISR\_AST - AST routine to respond to interrupt'

ROUTINE lp\_isr\_ast=

LP\_ISR\_AST, AST routine to handle an expected line printer interrupt

Functional description:

This AST routine gains control when the line printer generates an interrupt that the process expects. The interrupt usually indicates that the line printer has finished emptying its data buffers, and is ready to accept more input.

Since the process is hibernating, wake the process so that it can check line printer status, and continue outputting the data record.

Inputs:

4(AP) - an AST parameter (unused)

Outputs:

none

The routine preserves all registers.

Implicit outputs:

The sleeping process awakes.

--  
Wake(); ! Wake the sleeping process.

%SBTTL 'LP\_IODONE\_AST - AST routine at QIO-completion'

ROUTINE lp\_iodone\_ast=

++  
LP\_IODONE\_AST, An AST routine for QIO completion

Functional description:

This AST routine gains control when the connect to interrupt QIO request completes. This only occurs if VMS decides to disconnect the process from the interrupt in order to cancel I/O on the channel during process rundown. The action of the routine is to exit with a status indicating that the I/O was cancelled.

Inputs:

none

Outputs:

none

The routine preserves all registers.

--

\$exit( CODE=ss\$\_abort );

END ELUDOM



This image displays a grid of 100 small terminal window screenshots, arranged in 10 rows and 10 columns. Each window shows a different set of system commands and their outputs, demonstrating the capabilities of the VAX/VMS operating system. The windows are labeled with various system identifiers and commands, including:

- LPMULT B32
- DRMAST MAR
- ADDRIVER MAR
- TDRIVER MAR
- USSTEST MAR
- GBLSECURF MAR
- USSDISP MAR
- DOD\_ERAPAT MAR
- LBRMAC MAR
- XADDRIVER MAR
- LABLOCIN MAR
- DRSLV MAR
- DTE\_DF03 MAR
- SECRET MAR
- WORKO LIS
- EXAMPLES

The screenshots show a variety of system utilities, including file management, system configuration, and diagnostic tools. Each window typically displays a prompt, followed by a command and its corresponding output, which can include file listings, system status information, or error messages.