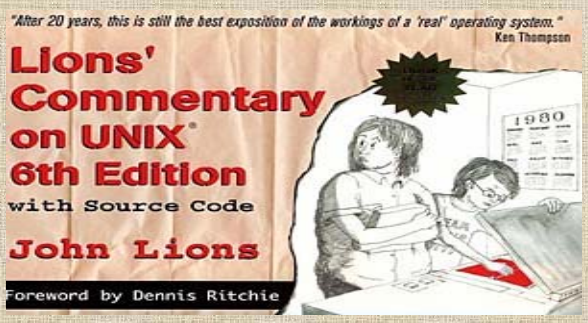


Chapter 9 and 10
Hardware Interrupts and Trap Routine

COP 5611

Presented by:
Aasavari Bhave, Manjula Babaladi

March 24, 2005

Outline

- Introduction
- Interrupt Vector and Trap Vector
- Flow of Control
- Interrupt Priority
- Rules for Interrupt Handlers
- Sources of Interrupts and Traps
- Assembler Routine
  - Trap
  - Clock Interrupt
  - System call
- Summary
- References

Introduction

- Hardware Interrupt :
  - Controllers of peripheral devices **interrupt** CPU for some operating system service.
  - This is caused by an event external to CPU.
  - Handled by a priority based scheme.
- Traps :
  - Result of unexpected internal CPU events like hardware or power failures.
  - A user mode program can explicitly use trap as part of a system call.
  - Ranked top priority.
- Both are essentially handled using similar software technique.

Device → Interrupt → Processor

- Set “interrupted gate” on the processor,
- Processor checks the gate between each instructions.
- If set, it invokes the Kernel entering mechanism.

Interrupt vector

Vector Location	Peripheral Device	Interrupt Priority	Process Priority
060	teletype input	4	4
064	teletype output	4	4
070	paper tape input	4	4
074	paper tape output	4	4
100	line clock	6	6
104	Programmable clock	6	6
200	line printer	4	4
220	RK disk drive	5	5

```
0502   br4  = 200
0503   br5  = 240
0504   br6  = 300
  / interrupt vector
0526   klin;    br4
0527   klou;    br4
0530   pcin;    br4
0531   pcou;  br6
0534   kwlp;    br6
0535   kwlp;    br6

0541   lpou;    br4
0544   rkio;    br5
```

Note: Interrupt priority and Process priority can be different

Trap Vector

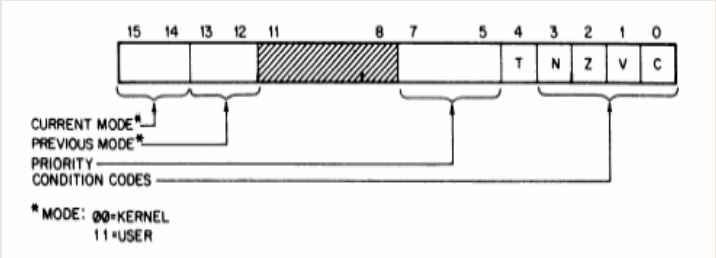
Vector Location	Trap type	Process Priority
004	Bus timeout	7
010	Illegal instruction	7
014	bpt-trace	7
020	iot	7
024	Power failure	7
030	Emulator trap	7
034	Trap instruction/ system entry	7
114	11/70 parity	7
240	Programmed interrupt	7
244	Floating point error	7
250	Segmentation violation	7

```
0505   br7 = 340
0511 /   trap vectors
0512   trap;   br7+0
0513   trap;   br7+1
0514   trap;   br7+2
0515   trap;   br7+3
0516   trap;   br7+4
0517   trap;   br7+5
0518   trap;   br7+6
0538   trap;   br7+7
0547   trap;   br7+7
0548   trap;   br7+8
0549   trap;   br7+9
```

File “low.s”

- At every Unix installation, a file “low.s” is generated by “mkconf” program which gives a list of actual peripherals present.
- Low.s has a call to 2 different entry points in the assembly code in file “m40.s” per interrupt or trap.
- The file “m40.s” is involved with handling interrupts and traps.

Processor Status Word



•Every process is associated with a Processor Priority.

•The processor priority for the interrupt handler is determined from the 7..5 bits of PSW.

Flow of Control

- Algorithm for handling interrupts

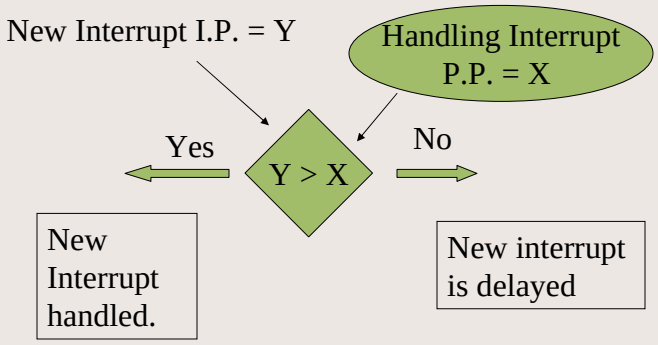
```
Input  : none
Output : none
{
        Save (push) current context layer;
                - CPU saves PSW and PC in the internal registers
        Determine interrupt source;
        Find interrupt vector;
                - PC and PSW are reloaded from vector location for the
                  event that causes the switch
                - push the internal registers into the newly created stack
        Call interrupt handler;
        Restore (pop) previous context layer;
                - "rtt" instruction reloads PC and PSW from the kernel
                  stack
}
```

Process Priority

- Every interrupt is associated with an interrupt priority which ranges from 0..7, 7 being the highest.
- Interrupt priority is determined by the hardware.
- The processor priority for any handler can be changed any time by O.S. but the interrupt priority is hard to change.
- Note - PDP11 has Unibus hardware which does not support interrupt priorities from 0..3.

Interrupt Priority Cnt'd

- Unix initialization for Interrupt Handler
  - Processor priority = Interrupt priority



Interrupt Priority Cnt'd

- If Processor priority < Interrupt priority, system is compelled to handle new interrupt of the same priority before completion of the current interrupt.
- For only the clock interrupt, the Processor priority is lower than the Interrupt priority as the next clock interrupt cannot wait for completion of the current clock interrupt.

Interrupt Priority Cnt'd

- During interrupt handling the processor priority may be raised to protect integrity of certain operations.
- While interrupt handler deals with a shared data structure, its priority is increased to 7 to avoid any interference.
- Similarly processor priority can be decreased using "spl" procedures [lines 1293..1315]

Rules for Interrupt Handlers

- System performance must not be degraded
  - Current interrupt cannot delay other interrupts excessively.
  - Current interrupt must not be preempted due to other interrupt frequently.
- Every process that is interrupted, must have a mechanism to wake up the process which was waiting on that interrupt.

Rules for Interrupt Handlers Cnt'd

- The handler should make references to the "u" structure in the process waiting for it rather than "u" structure in current process.
- Interrupt handler should not call sleep(), as the process thus suspended, will be prevented to continue its execution.

Sources of Interrupts/Traps

- "main" calls "fuibyte" or "fuiword" repeatedly until a negative value is returned. This value is returned in r0.
- Clock generates an interrupt every clock tick.
- Process #1 is about to execute a "trap" instruction as part of the system call on "exec".

“fuiword” routine

```
0845  _fuiword
0846   mov  2(sp), r1 // fuiword argument on
                       // the stack moved to r1

0847   _fuword
0848   jsr   pc, gword  // call to gword
0849   rts   pc
0850
0851  gword:
0852  mov  PS, -(sp)   // PSW saved on stack
0853  bis    $340, PS   // priority = 7
0854  mov  nofault, -(sp)

0855  mov  $err, nofault
0856  mfpi (r1)          // fetch word from user space
```

sp →
sp →
sp →
sp →
sp →

Return main
r1
Return to 0849
PSW
Nofault address
r0 = mfpi return value

fuiword normal return

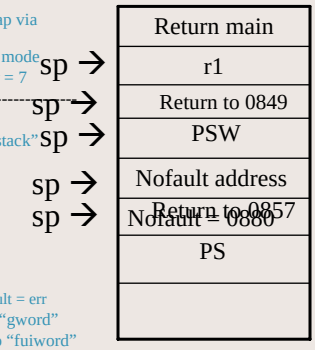
```
0857   mov     (sp)+, r0      // the value is transferred from the stack to r0
0858   br        1f

0875  1 :
0876   mov    (sp)+, nofault // the previous values of “nofault”
0877   mov   (sp)+, PS        // and PS are  restored
0878   rts      pc              // return via line 0849
```

sp →

Return main
r1
Return 0849
PSW
Nofault address
mfpi return value

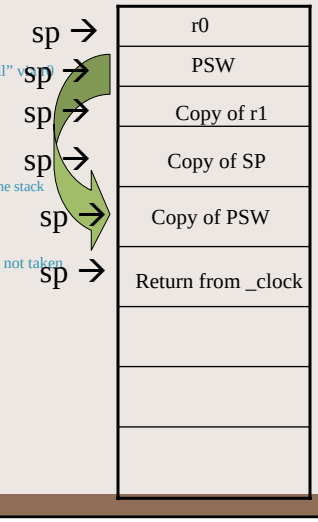
fuiword abort

```
0856  mfpi    (r1)          // mfpi instruction aborted. PC = 0857, trap via
                            // vector location 4 will occur
0512  trap; br7 + 0         //New PC = 0512, Present mode = kernel mode
                            // Previous mode = kernel mode , Priority = 7
//-------------------------------------------------------------------------
0755  trap:
0756  mov   PS, -4(sp)      // Save  PSW beyond the current "top of stack"
0757  tst   nofault         // "nofault" <- err non zero
0758  bne   1f
0759  mov   SSR0, sst
0760  mov   SSR2, ssr + 4
0761  mov   $11, SSR0
0762  jsr0, call;  _trap
0763    / no return
0764  1:
0765  mov   $1, SSR0        // Reinitalise the MMU
0766  mov   nofault, (sp)   // overwrites return addr of gword with nofault = err
0767  rtt                   // returns to first word of "err" and not to "gword"
0880  err:          // restores "nofault" and PS. Skips the return to "fuiword"
0881  mov   (sp)+, nofault                                              /
0882  mov   (sp)+, PS                                                   /
0883  tst   (sp)+                                                       /
0884  mov   $-1, r0     // r0 -> -1 and returns directly to the calling routine
0885  rts   pc
0849  rts   pc
```



Clock Interrupt

```
PC    =   address of location labeled "kwlp"(0568)
PSW ->    present mode = kernel mode
          previous mode = kernel or user mode
          Priority =6
0570      kwlp: jsr  r0, call;   _clock  // This instruction is a subroutine "call" v
          r0 = address of *( _clock()) in clock.c.
0776      call:
0777      mov PS, -(sp)         // Copy PS onto the stack
0778      1:
0779      mov r1, -(sp)         // copy r1 onto the stack
0780      mfpi  sp              // copy SP for previous user address space onto the stack
0781      mov 4(sp), -(sp)      // Copy the  copy of PS onto the stack
0782      bic   $!37, (sp)      // Mask all but lower 5 bits of PSW.
0783      bit   $30000, PS      // Test if the previous mode is kernel or user
0784      beq  1f               // If Previous mode is kernel mode, branch is not taken

        IF Previous Mode = Kernel Mode :
0798      bis   $30000, PS      // set previous mode = user mode
0799      jsr   pc, *(r0)+      // call to subroutine _clock in clock.s - 3725
0800      cmp (sp)+, (sp)+      // PSW and copy of SP deleted
0801      2:
0802      mov (sp)+, r1         // Restore r1
0803      tst   (sp)+
0804      mov (sp)+, r0         // restore r0
0805      rtt                   // return to previous kernel mode routine.
```



Clock Interrupt Cnt'd

If Previous Mode = User Mode

```
0785    jsr pc , *(r0)+     // call to  _clock in clock.c
0786    2:
0787    bis $340,PS         // priority = 7
0788    tstb _runrun        //checks to see if a higher priority process is ready to run
0789    beq 2f
0790     bic $340,PS        // priority = 0
0791    jsr pc,_swtch       //Allow higher priority process to proceed
0792    br 2b               // repeat the test
```

User Program Traps

```
0518   trap; br7+6.          // PSW = br7 + 6, PC = trap

0755  trap:                                          sp →
0756  mov   PS, -4(sp)      // Save PSW to stack     sp →
0757  tst     nofault       //nofault = 0, branch not taken
0758  bne     1f                                     sp →
0759  mov  SSR0, ssr        //memory management status stored
0760  mov  SSR2, ssr + 4
0761  mov   $1, SSR0
0762  jsr     r0, call1;  _trap // save r0, pc = call1 r0 = address of
                                // memory location that contains "_trap"

0771  call1:
0772  tst     -(sp)         // SP adjust to point to location copy of PS
0773  bic     $340, PS      // CPU priority = 0
0774  br      1f            // branch to second instruction of  "call"
```

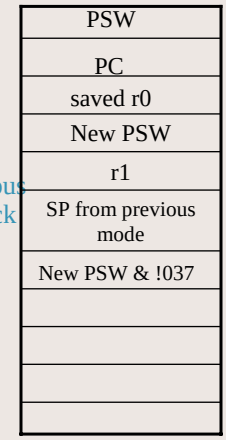
PSW
PC
saved r0
New PSW

User Program Traps Cnt'd

```
// Code shared with interrupt processing
0776  call:
0777  mov   PS, -(sp)
0778  1:
0779  mov   r1, -(sp)
0780  mfpi  sp        // Copy the SP for the previous
                      // address space onto the stack
0781  mov   4(sp), -(sp)
0782  bic   $!37, (sp)   // mask new PSW

0783  bit   $30000, PS
0784  beq   1f
0785  jsr   pc, *(r0)+
```

sp →

sp →



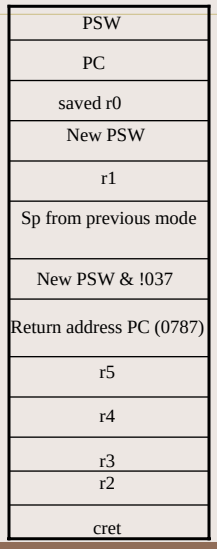
User Program Traps Cnt'd

```
jsr r5, csv
        1421  mov r5, r0
        1422  mov sp, r5
        1423  mov r4, -(sp)
        1424  mov r3, -(sp)
        1425  mov r2, -(sp)
        1426  jsr pc, (r0)

2693  trap(dev, sp, r1, nps, r0, pc, ps)

2754  callp = &sysent[fuiword(pc-2) &077];
        // Kernel retrieves bottom 6 bits  of the word that
        contains user trap instruction and uses as index into
        sysent

0787  bis  $340, PS     // Kernel returns and checks if
                        // other thread should run
0794  tst  (sp)+        // remove saved new PSW & !037
```

sp →

sp →

sp →



Summary

- Hardware interrupts are events caused by peripheral devices.
- Traps are highest priority interrupts caused by hardware failure or explicit system calls in user programs.
- Hardware interrupts and traps are handled with similar mechanism by Unix.
- Interrupts / Traps handling leads to saving context, PC + PSW on the active stack and start the interrupt process routine and subsequently involves retrieving back the PC + PSW before process resumes.
- Processor priority of interrupt handlers is increased at runtime to preserve operational integrity.
- Processor priority for clock interrupts can be decreased for handling frequent clock interrupts.
- Low.s lists all the hardware interrupts and trap types and has calls to code 0777 -0805.
- The code 0755 – 0805 in "m40.s" handles the hardware and trap and routes the interrupt / trap to be processed by the particular service routine.
- Knowledge of assembly and stack operation is extremely useful in understanding interrupt and trap handling.

References

- John Lions,"*Chapter 1..10, Lions' Commentary on Unix, 6th edition*"
- M. J. Bach. "*The Design of the UNIX Operating System*" Prentice-Hall, 1987
- MIT Open Course Ware , http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6828Fall2003/LectureNotes/detail/lec7.htm
- "*Processor Handbook, PDP 11/40"*, Copyright @1972, DEC.


Thank You