Interrupts

Purpose

• Events happen outside control of program
  – Need to respond to those events
• Interrupt provides notice of event
• Two types:
  – Hardware – external event
  – Software – program generates event
Hardware Interrupt

- External device generated
  - Causes signal on CPU pin
- CPU-generated
  - E.g., divide by zero
- May have information describing interrupt
  - Interrupt Vector
  - Interrupt Priority Level

Software Interrupt

- Special instruction
  - May have argument as “parameter”
- Acts like a subroutine call
  - Saves more state
  - May change priorities/protections
  - Can transfer outside of program’s virtual memory
- On SPARC, “Trap” instruction
Interrupt Processing

- Save state
  - Disable interrupts
  - Program Counter
  - Condition Codes
  - Registers?
- Transfer to interrupt handler
  - Fixed location
  - Vector based on IRQ/IPL
- Identify source of interrupt
  - Contained in IRQ/IPL
  - Poll devices
- Process interrupt
- Restore state

<table>
<thead>
<tr>
<th>Memory: Interrupt Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt 1 address</td>
</tr>
<tr>
<td>Interrupt 2 address</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Interrupt 1 handler</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Program</td>
</tr>
</tbody>
</table>

Terminology and Connotations
*(Following adapted from Blaauw and Brooks)*

- asynchronous, external
  - interrupt - external device (I/O, clock, etc.)
  - machine check - hardware error or failure
- synchronous, internal
  - general terms
    - alarm
    - exception - special or undefined case (e.g., divide by zero)
    - internal interrupt
    - program check
    - trap (sprung like a "mousetrap")
  - terms associated with no continued execution
    - abort - exception in the middle of an instruction
  - terms associated with continued execution (either restart or resume)
    - fault - missing information (e.g., page fault)
  - terms associated with user request to OS
    - mmf (master mode entry)
    - software interrupt (e.g., INT opcode in x86)
    - svc (supervisor call)
    - syscall
    - trap (trap always on SPARC)
Entry point

- fixed memory address (e.g., PDP-8)
  - poll to find cause (e.g., PDP-8)
  - code indicating cause placed in register
- one of many memory locations (interrupt vector)
  - number of vectors
    - single, globally shared (e.g., Electrologica X-1, S/360, x86)
    - multiple, one per process (e.g., IBM Stretch)
  - vector contents
    - interrupt vector entry holds new PC (and maybe new PSW)
      (e.g., S/360, x86)
    - interrupt vector holds blocks of instructions (e.g., SPARC)
- different PC (e.g., DYSEAC, TX-2)

Return Linkage

- fixed memory location(s) (e.g., PDP-8, S/360)
- memory stack (e.g., x86, M68K)
- register (e.g., SPARC)
Permission

- single interrupt enable bit (e.g., PDP-8, 8086 w/o PIC)
  - enable/disable interrupt instructions
- interrupt enable mask in PSW (e.g., S/360)
- interrupt priority code in PSW (e.g., SPARC)
- nonmaskable (cannot be ignored)
- breakpoint bit in each instruction (e.g., DYSEAC)
  - device priority along with break and dismiss bit in each instructions (e.g., TX-2)

Optimizations

- cause determination (see cause register and interrupt vector)
- additional register sets
- interrupt poll to coalesce multiple interrupts (e.g., ITI on B5500 in 1960s, test pending interrupt on S/370 XA)
Concepts and Terminology

- **Signal**: terminology for interrupt
  - Multiple types (64 on SSLab machines)
- **Sources**:  
  - Errors (divide by zero, invalid address)  
  - External Events (devices)  
  - Explicit requests  
    - Raise: interrupt self  
    - Kill: interrupt another process
Concepts and Terminology (cont.)

- Synchronous events happen as a result of the process that processes the signal
  - Raise
  - Errors
  - Processed immediately
- Asynchronous events generated externally
  - External events
  - Kill
  - Become “pending”
- (most) signals can be blocked
  - Become pending until unblocked

Handling Interrupts

- Signal Handler: Procedure to handle an interrupt
  - void handler(int signum) {
      /* Procedure to do whatever should be done */
  }
  - Saving state, etc. handled automatically
- Be careful if accessing global variables / data structures
  - sig_atomic_t – integer type that won’t be interrupted mid-access
  - Block signals before accessing data structure that signal handler touches
Handling Interrupts (cont.)

• Setting up the “interrupt vector”
  – signalhandler_t old_handler = signal(SIGSEGV, handler)

• Default handlers:
  – SIG_DFL – default for the specified signal
  – SIG_IGN – ignore the specified signal

• Default handler reinstated when signal occurs
  – Call signal before returning to reinstate your own handler

Handling Errors
setjmp and longjmp

• What if you don’t want to return
  – E.g., handling divide by zero error

• Could exit (best: raise same signal)
  – void handler(int sig) { raise(sig); }

• Could also go elsewhere
  – jmp_buf restart;
    main() { if (setjmp(restart)) /* arriving from interrupt */
      else /* normal program execution */ }
  – In signal handler:
    longjmp(restart, -1);
Blocking signals

- sigprocmask (SIG_BLOCK, &signal_set, &old_set);
  - Also SIG_UNBLOCK, SIG_SETMASK

- Setting sigset_t signal_set
  - sigemptyset(&signal_set);
  - sigaddset(&signal_set, SIGKILL);

Linux signals

1) SIGHUP  2) INT  3) SIGQUIT  4) SIGILL
5) SIGTRAP  6) SIGABRT  7) SIGBUS  8) SIGFPE
9) SIGKILL  10) SIGUSR1  11) SIGSEGV  12) SIGUSR2
13) SIGPIPE  14) SIGALRM  15) SIGTERM  16) SIGSTKFLT
17) SIGCHLD  18) SIGCONT  19) SIGSTOP  20) SIGTSTP
21) SIGTTIN  22) SIGTTOU  23) SIGURG  24) SIGXCPU
25) SIGXFSZ  26) SIGVTALRM  27) SIGPROF  28) SIGWINCH
29) SIGIO  30) SIGPWR  31) SIGSYS  32) SIGRTMIN
33) SIGRTMIN+1  34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4
37) SIGRTMIN+5  38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8
41) SIGRTMIN+9  42) SIGRTMIN+10  43) SIGRTMIN+11  44) SIGRTMIN+12
45) SIGRTMIN+13  46) SIGRTMIN+14  47) SIGRTMIN+15  48) SIGRTMIN+16
49) SIGRTMIN+17  50) SIGRTMIN+18  51) SIGRTMIN+19  52) SIGRTMIN+20
53) SIGRTMIN+21  54) SIGRTMIN+22  55) SIGRTMIN+23  56) SIGRTMIN+24
57) SIGRTMIN+25  58) SIGRTMIN+26  59) SIGRTMIN+27  60) SIGRTMIN+28
61) SIGRTMIN+29  62) SIGRTMIN+30  63) SIGRTMIN+31  64) SIGRTMAX
Things to Remember

- Signal handler / return automatic
  - setjmp/longjmp to return elsewhere
- Reset when signal occurs
  - Call signal again if you want to keep handling
- Be careful when updating data structures
  - Signal may have occurred in the middle of something
  - Block/unblock around “uninterruptable” code

Want to try it?

- Option 1: We’ll see if we can build an example on-the-fly
- Option 2: Continue with Chapter 16 (more on device drivers)