Lecture 17: Exceptions

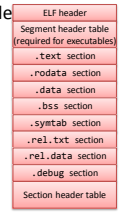
Computer Architecture and
Systems Programming
(252-0061-00)

Timothy Roscoe

Herbstsemester 2012

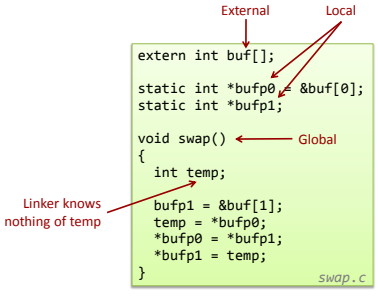
Last Time

- Optimization for the memory hierarchy
- Linking
  - Symbol resolution: Associate each symbol reference with exactly one definition
  - Relocation: Merge all `.o` files into one executable
- Object files
  - Relocatable object files (`.o`)
  - Executables
  - Shared object file (`.so`)
  - ELF format

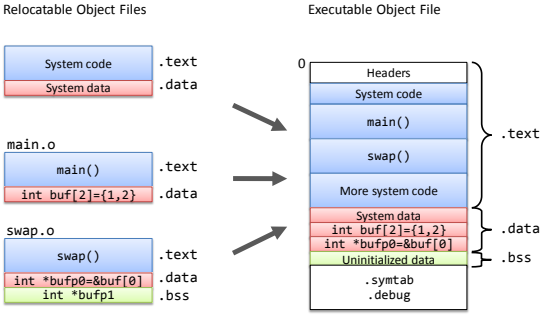


Last Time: Symbol Resolution

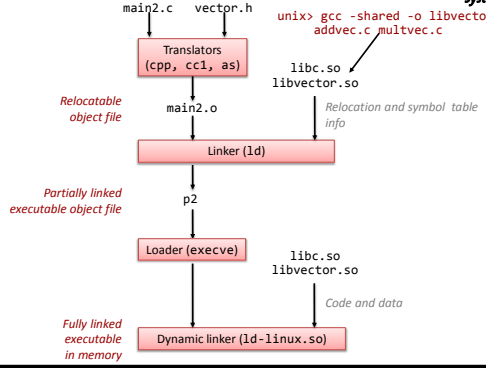
- Symbols
  - Global
  - External
  - Local
- Symbols
  - Strong
  - Weak



Last time: Relocation



Dynamic linking at load-time

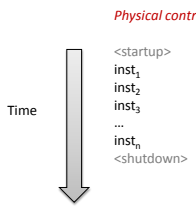


Today: Exceptions

- Exceptional Control Flow
- Exception vectors
- Kernel mode
- Synchronous vs. Asynchronous
- Traps
- Page faults
- Invalid references

Control Flow

- Processors do only one thing:
  - From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
  - This sequence is the CPU's *control flow* (or *flow of control*)



Altering the Control Flow

- Up to now: two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return

  Both react to changes in ***program state***
- Insufficient for a useful system:
  Difficult to react to changes in ***system state***
  - data arrives from a disk or a network adapter
  - instruction divides by zero
  - user hits Ctrl-C at the keyboard
  - System timer expires
- System needs mechanisms for "exceptional control flow"

Exceptional Control Flow

- Exists at all levels of a computer system
- Low level mechanisms
  - Exceptions
    - change in control flow in response to a system event (i.e., change in system state)
  - Combination of hardware and OS software
- Higher level mechanisms
  - Process context switch
  - Signals
  - Nonlocal jumps: `setjmp()/longjmp()`
  - Implemented by either:
    - OS software (context switch and signals)
    - C language runtime library (nonlocal jumps)
  - Language-level exceptions (Java, etc.)

Exceptions

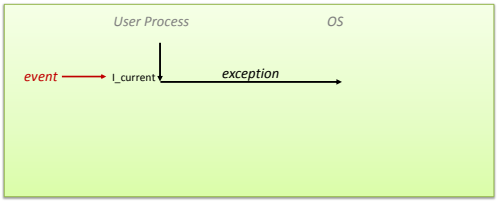
- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)

Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)

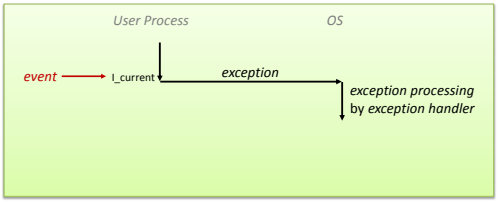
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



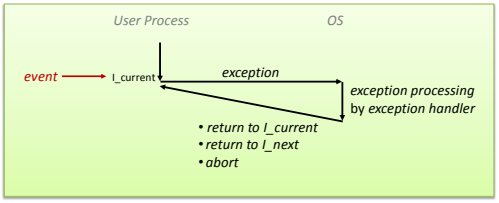
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



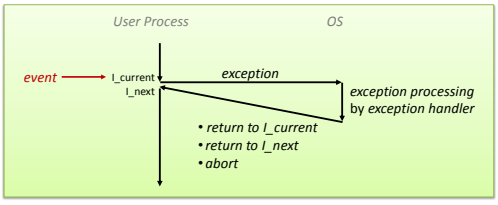
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



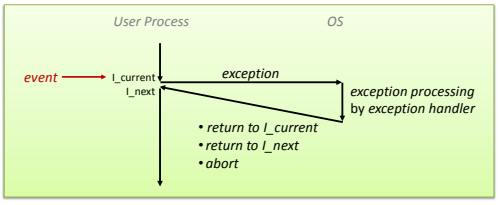
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



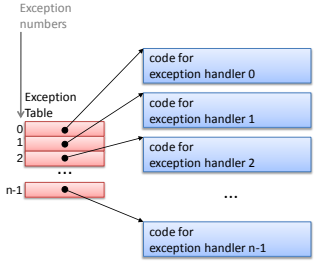
Exceptions

- An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



- Examples:
  div by 0, arithmetic overflow, page fault, I/O request completes, Ctrl-C

Interrupt Vectors



- Each type of event has a unique exception number k
- k = index into exception table (a.k.a. interrupt vector)
- Handler k is called each time exception k occurs

x86 Exception Vectors

0	Divide error
1	Debug exception
2	**Non-maskable interrupt (NMI)**
3	Breakpoint
4	Overflow
5	Bounds check
6	Invalid opcode
7	Coprocessor not available
8	Double fault
9	Coprocessor segment overrun (386 or earlier)
A	Invalid task state segment
B	Segment not present
C	Stack fault
D	General protection fault
E	Page Fault
F	Reserved
10	Math fault
11	Alignment check
12	Machine check
13	SIMD floating point exception
14-1F	Reserved to Intel
20-FF	**Available for external interrupts**

Check pp. 183:
http://download.intel.com/design/processor/manuals/253665.pdf

Kernel mode

- Exceptions cause a switch to ***kernel mode***
  - Also: supervisor mode, privileged mode, ring 0, etc.
- Things are very different:
  - Access to system state (virtual memory, etc.)
  - Some exceptions are disabled
  - Some new instructions and registers
- Details vary between processors
  - Even in the same architecture

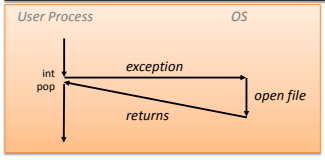
Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - ***Traps***
    - Intentional
    - Examples: ***system calls***, breakpoint traps, special instructions
    - Returns control to "next" instruction
  - ***Faults***
    - Unintentional but possibly recoverable
    - Examples: page faults (recoverable), protection faults (unrecoverable), floating point exceptions
    - Either re-executes faulting ("current") instruction or aborts
  - ***Aborts***
    - unintentional and unrecoverable
    - Examples: parity error, machine check
    - Aborts current program

Trap Example: Opening File

- User calls: `open(filename, options)`
- Function open executes system call instruction `int`

```
0804d070 <__libc_open>:
 . . .
 804d082: cd 80                  int    $0x80
 804d084: 5b                     pop    %ebx
 . . .
```

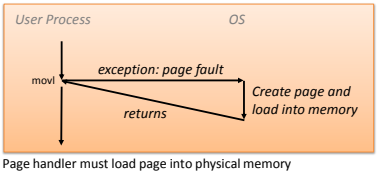


- OS must find or create file, get it ready for reading or writing
- Returns integer file descriptor

Fault Example: Page Fault

- User writes to memory location
- That portion (page) of user's memory is currently on disk

```
80483b7: c7 05 10 9d 04 08 0d       movl   $0xd,0x8049d10
```



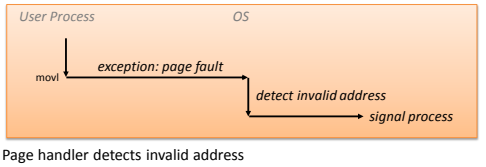

```
int a[1000];
main ()
{
    a[500] = 13;
}
```

- Page handler must load page into physical memory
- Returns to faulting instruction
- Successful on second try

Fault Example: Invalid Memory Reference

```
int a[1000];
main ()
{
    a[5000] = 13;
}
```

```
80483b7:        c7 05 60 e3 04 08 0d     movl   $0xd,0x804e360
```



- Page handler detects invalid address
- Sends SIGSEGV signal to user process
- User process exits with "segmentation fault"

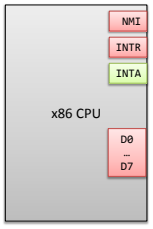
Asynchronous Exceptions (Interrupts)

- Caused by events external to the processor
  - Indicated by setting the processor's interrupt pin
  - Handler returns to "next" instruction
- Examples:
  - I/O interrupts
    - hitting Ctrl-C at the keyboard
    - arrival of a packet from a network
    - arrival of data from a disk
  - Hard reset interrupt
    - hitting the reset button
  - Soft reset interrupt
    - hitting Ctrl-Alt-Delete on a PC

Interrupts

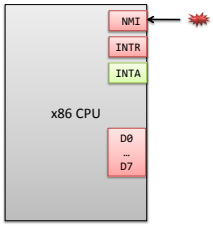
- CPU **Interrupt-request line** triggered by I/O device
  - Might be edge- or level-triggered
- **Interrupt handler** receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some **nonmaskable**
- Interrupt mechanism also used for exceptions

Basic x86 Interrupts



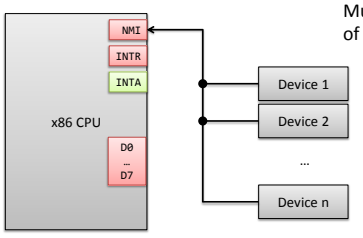
- Two interrupt **pins**:
  1. INTR: interrupt request
  2. NMI: non-maskable interrupt
- Fairly typical even on other architectures
  - E.g. ARM, etc.

x86 Interrupts: NMI



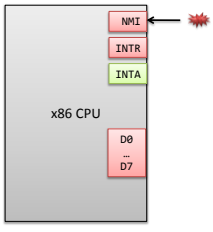
- NMI asserted ⇒
  - CPU completes current instruction
  - Issues Exception #2
  - Always.
- Cannot be disabled by the processor.
- Reserved for
  - Major hardware faults, e.g. memory parity error
  - "Watchdog" timer

x86 Interrupts: NMI



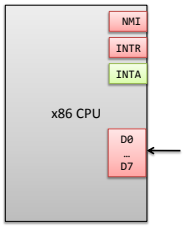
Multiple sources of interrupts?

x86 Interrupts: NMI



- What caused the NMI?
- No obvious way to tell: OS (`INT 0x2` handler) must poll potential sources
- For normal use: plenty of devices ⇒ slow reaction, inefficient
- Might get a second NMI while first is still being handled!

x86 Interrupts: IRQ



- Interrupt request ⇒
  - Can be disabled by `IE` status flag (`CLI/STI`)
  - If enabled, complete current instruction, then:
- CPU acknowledges using INTA pin
- Interrupt vector is then supplied on the data bus
- CPU issues exception # from the vector

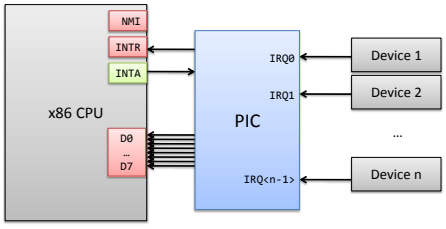
x86 Exception Vectors

0	Divide error
1	Debug exception
2	**Non-maskable interrupt (NMI)**
3	Breakpoint
4	Overflow
5	Bounds check
6	Invalid opcode
7	Coprocessor not available
8	Double fault
9	Coprocessor segment overrun (386 or earlier)
A	Invalid task state segment
B	Segment not present
C	Stack fault
D	General protection fault
E	Page Fault
F	Reserved
10	Math fault
11	Alignment check
12	Machine check
13	SIMD floating point exception
14-1F	Reserved to Intel
20-FF	**Available for external interrupts**

Remaining problems

- What caused the interrupt?
  - Can tell from the vector, but…
- How does the device know which vector to give?
  - Most early devices just had a single pin (for IRQ)
  - Vector could only be changed using switches or jumpers
  - Led to "interrupt conflicts" in early PCs
- What happens when more than one interrupt happens?
  - Should not miss interrupts – bad!

Programmable Interrupt Controllers



Programmable Interrupt Controllers

- Map physical interrupt **pins** (from device, slot, socket, etc.) to interrupt **vectors**
  - Mapping picked by the OS ⇒ flexible

Programmable Interrupt Controllers

- Map physical interrupt **pins** (from device, slot, socket, etc.) to interrupt **vectors**
  - Mapping picked by the OS ⇒ flexible
- **Buffer** simultaneous interrupts
  - Deliver each interrupt vector separately
  - Won't lose some device's interrupt

Programmable Interrupt Controllers

- Map physical interrupt **pins** (from device, slot, socket, etc.) to interrupt **vectors**
  - Mapping picked by the OS ⇒ flexible
- **Buffer** simultaneous interrupts
  - Deliver each interrupt vector separately
  - Won't lose some device's interrupt
- **Prioritize** interrupts
  - Allow some device IRQs to interrupt others'
  - Again, selected by the OS

Programmable Interrupt Controllers

- Map physical interrupt **pins** (from device, slot, socket, etc.) to interrupt **vectors**
  - Mapping picked by the OS ⇒ flexible
- **Buffer** simultaneous interrupts
  - Deliver each interrupt vector separately
  - Won't lose some device's interrupt
- **Prioritize** interrupts
  - Allow some device IRQs to interrupt others'
  - Again, selected by the OS
- Selectively **mask** any individual device's interrupts
  - Useful for high-interrupt rate devices
  - Useful at boot!

Modern PICs

- Every processor in a modern PC has a "Local APIC"
  - "Local Advanced Programmable Interrupt Controller"
  - PIC → APIC → LAPIC → xAPIC → x2APIC → …
- Additional capabilities:
  - Inter-processor interrupts
  - Programmable timer
  - Sophisticated interrupt scheduling
  - Etc.
- Integrated onto same die as the core itself
- Also: IOxAPIC
  - Off-chip, attached to device I/O interconnect

Summary

- Exceptions
  - Events that require nonstandard control flow
  - Generated externally (interrupts) or internally (traps and faults)
- We'll more of these when we look at:
  - Device handling
  - Virtual memory
  - Multiprocessors
  - Operating Systems