Exceptional Control Flow: Exceptions and Processes

15-213 : Introduction to Computer Systems
14th Lecture, October 12th, 2017

Instructor:

Randy Bryant

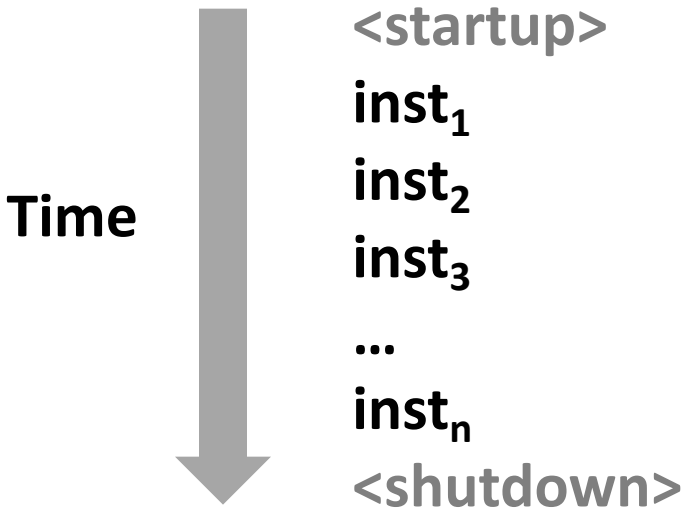
Today

- **Exceptional Control Flow**
- **Exceptions**

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*)

Physical control flow



Altering the Control Flow

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

  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

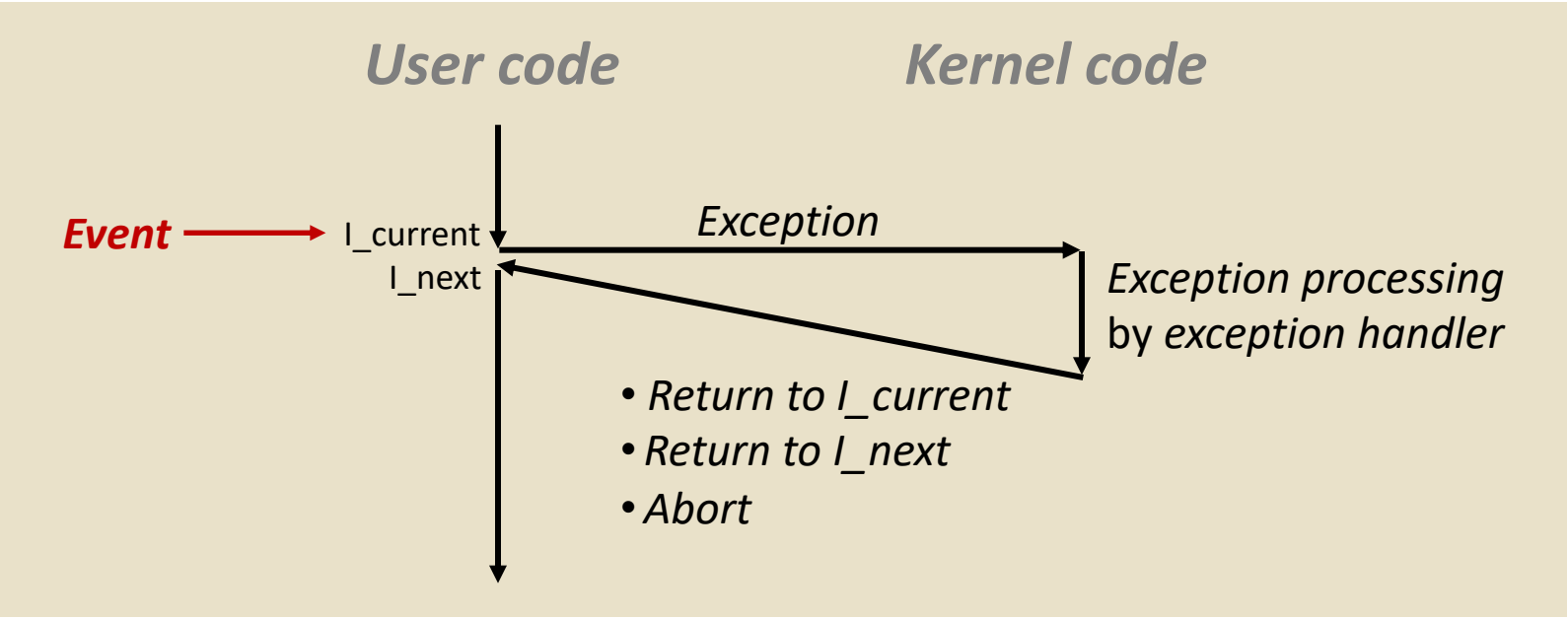
- **Exists at all levels of a computer system**
- **Low level mechanisms**
  - 1. **Exceptions**
    - Change in control flow in response to a system event (i.e., change in system state)
    - Implemented using combination of hardware and OS software
- **Higher level mechanisms**
  - 2. **Process context switch**
    - Implemented by OS software and hardware timer
  - 3. **Signals**
    - Implemented by OS software
  - 4. **Nonlocal jumps**: `setjmp()` and `longjmp()`
    - Implemented by C runtime library

Today

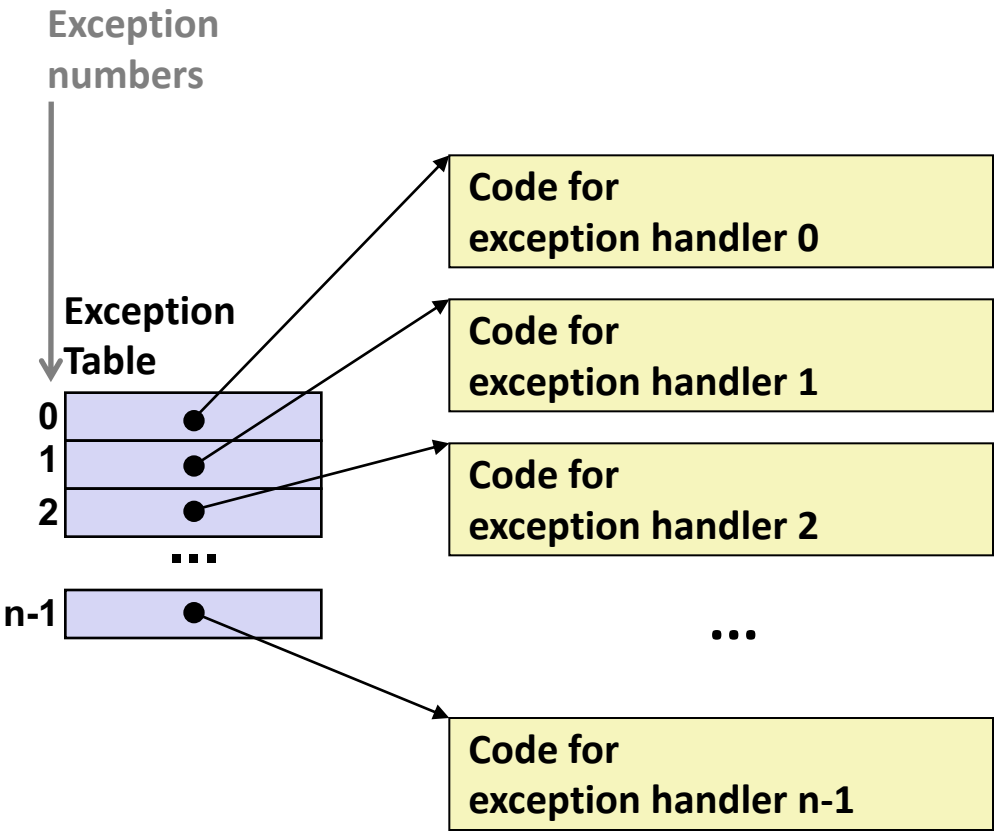
- Exceptional Control Flow
- **Exceptions**

Exceptions

- **An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)**
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C

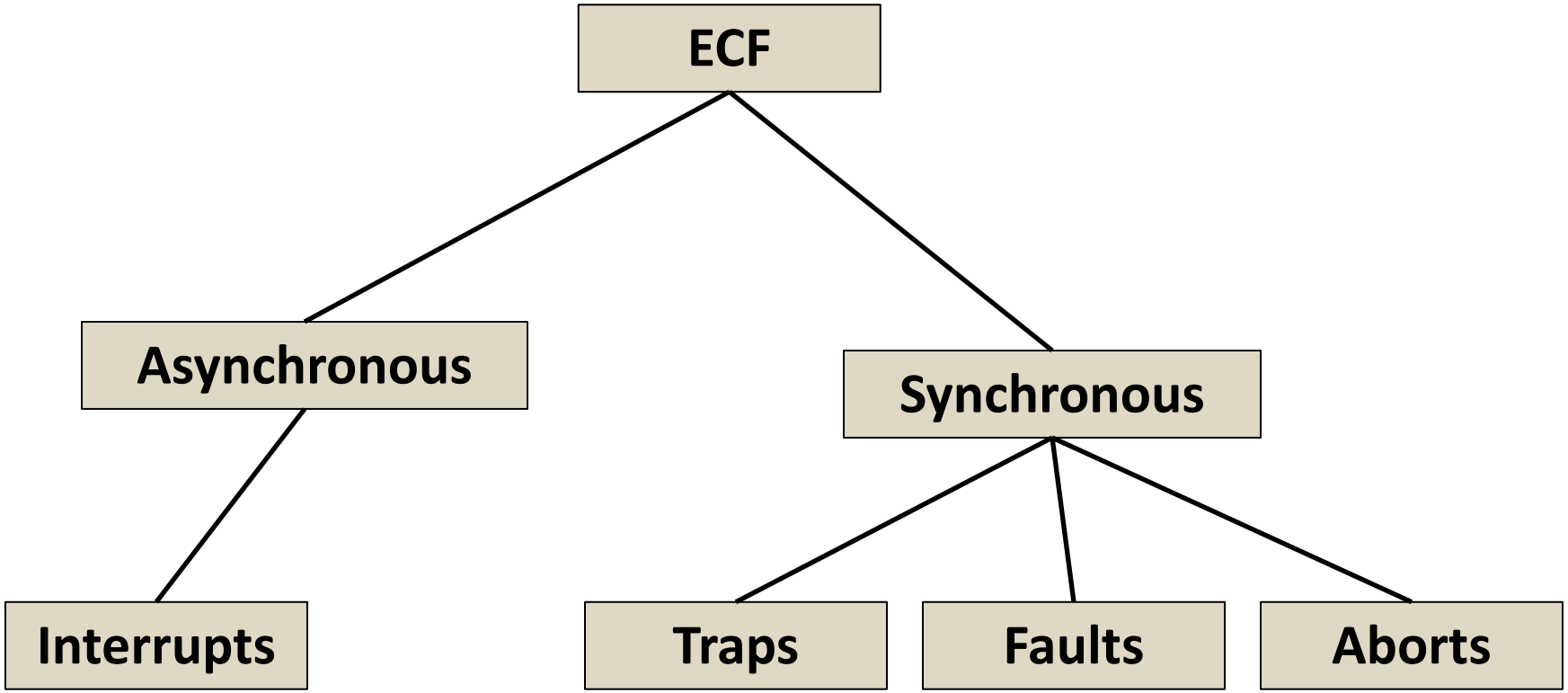


Exception Tables



- 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

(partial) Taxonomy



Asynchronous Exceptions (Interrupts)

- **Caused by events external to the processor**
  - Indicated by setting the processor's *interrupt pin*
  - Handler returns to "next" instruction

- **Examples:**
  - **Timer interrupt**
    - Every few ms, an external timer chip triggers an interrupt
    - Used by the kernel to take back control from user programs
  - **I/O interrupt from external device**
    - Hitting Ctrl-C at the keyboard
    - Arrival of a packet from a network
    - Arrival of data from a disk

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: illegal instruction, parity error, machine check
    - Aborts current program

System Calls

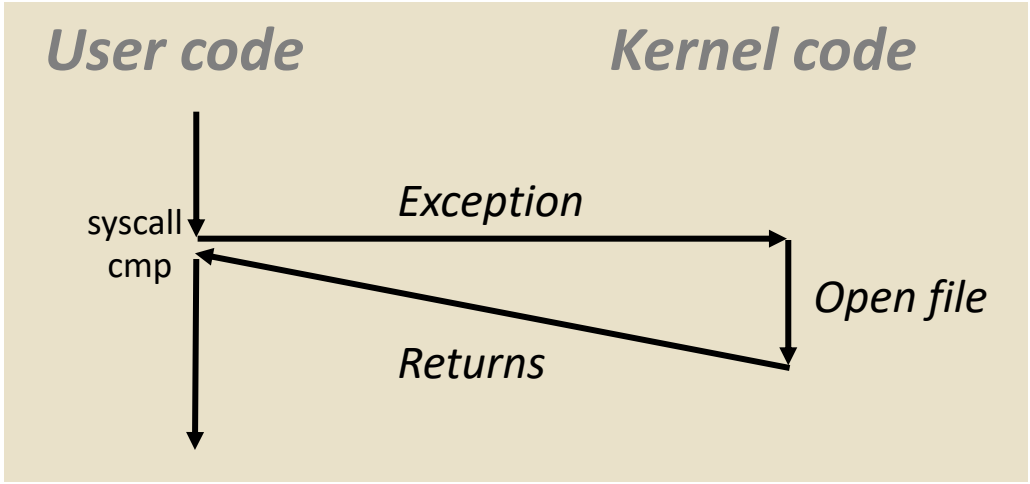
- **Each x86-64 system call has a unique ID number**
- **Examples:**

Number	*Name*	*Description*
0	`read`	Read file
1	`write`	Write file
2	`open`	Open file
3	`close`	Close file
4	`stat`	Get info about file
57	`fork`	Create process
59	`execve`	Execute a program
60	`_exit`	Terminate process
62	`kill`	Send signal to process

System Call Example: Opening File

- User calls: **`open(filename, options)`**
- Calls **`__open`** function, which invokes system call instruction **`syscall`**

```
0000000000e5d70 <__open>:
...
e5d79:   b8 02 00 00 00      mov  $0x2,%eax  # open is syscall #2
e5d7e:   0f 05               syscall         # Return value in
%rax
e5d80:   48 3d 01 f0 ff ff   cmp  $0xfffffffffffff001,%rax
...
e5dfa:   c3                  retq
```



- `%rax` contains syscall number
- Other arguments in `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9`
- Return value in `%rax`
- Negative value is an error corresponding to negative `errno`

System Call I

- User calls: **open(f**
- Calls **__open** functi

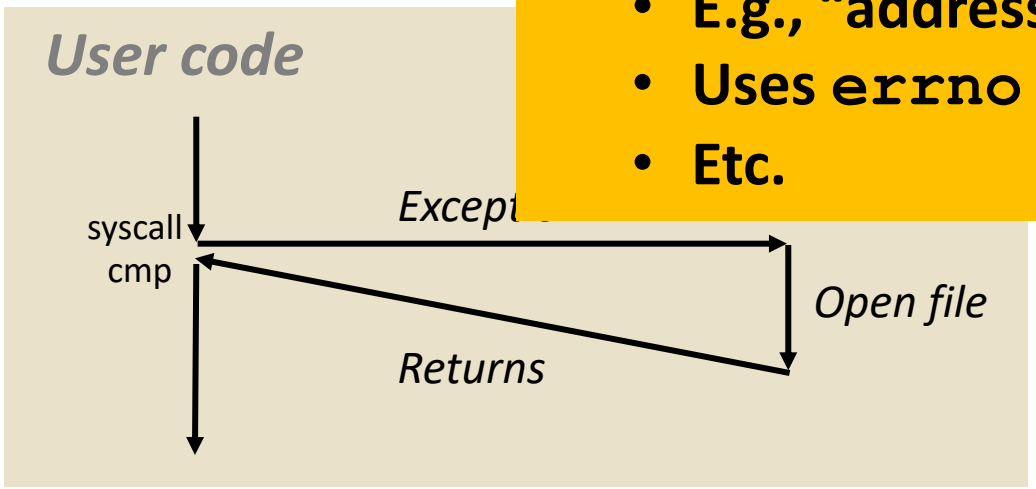
```
0000000000e5d70 <
...
e5d79:   b8 02 00
e5d7e:   0f 05
%rax
e5d80:   48 3d 01
...
e5dfa:   c3
```

Almost like a function call

- **Transfer of control**
- **On return, executes next instruction**
- **Passes arguments using calling convention**
- **Gets result in `%rax`**

One Important exception!

- **Executed by Kernel**
- **Different set of privileges**
- **And other differences:**
  - **E.g., "address" of "function" is in `%rax`**
  - **Uses `errno`**
  - **Etc.**



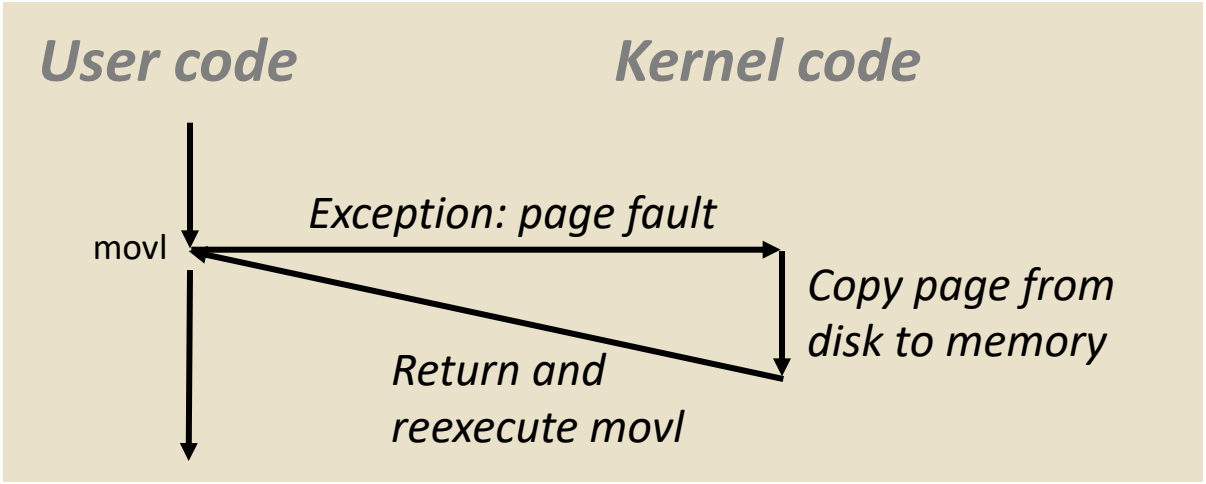
- Return value in `%rax`
- Negative value is an error corresponding to negative `errno`

Fault Example: Page Fault

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

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

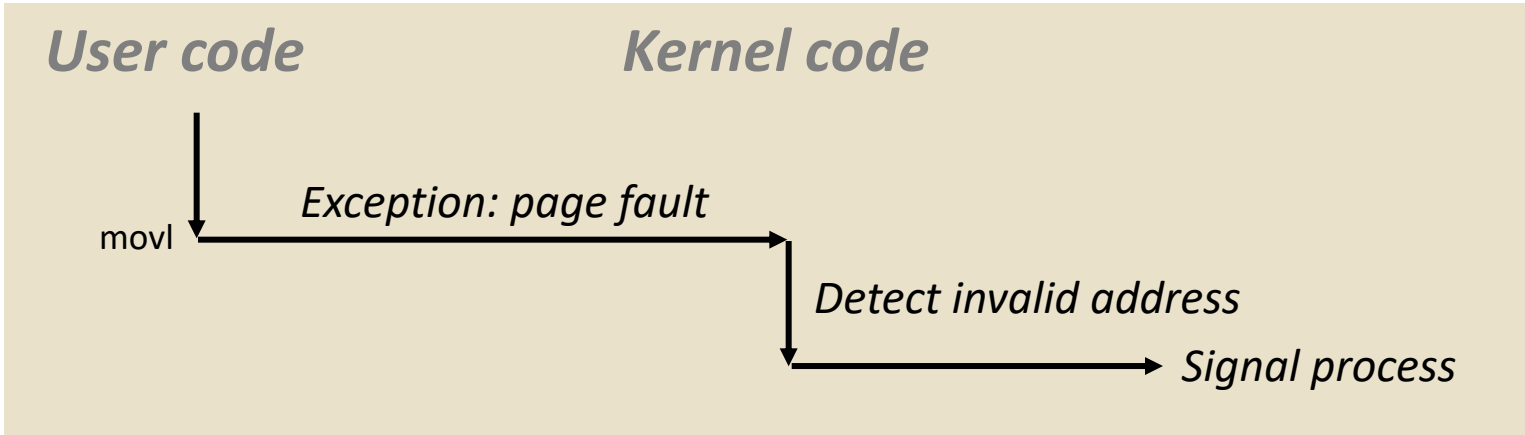
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



Fault Example: Invalid Memory Reference

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

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



- Sends **SIGSEGV** signal to user process
- User process exits with "segmentation fault"

Summary

- **Exceptions**
  - Events that require nonstandard control flow
  - Generated externally (interrupts) or internally (traps and faults)