Imperial College London

113: Architecture

Spring 2018

Lecture: Exceptional Control Flow

Instructor: Dr. Jana Giceva

Today: System Control Flow

- **Control Flow**
- **Exceptional Control Flow**
- Asynchronous exceptions (Interrupts)
- Synchronous exceptions (Traps and Faults)

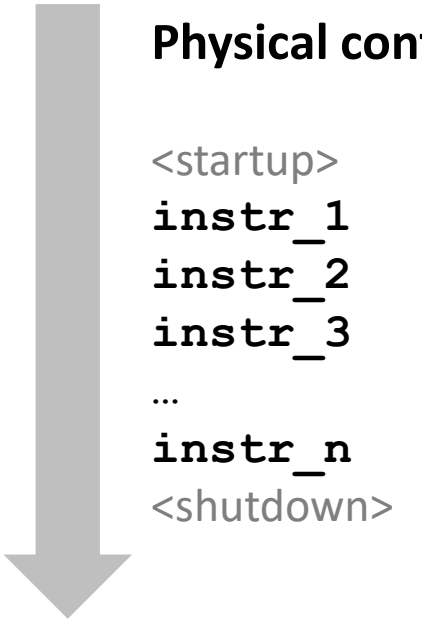
Control Flow

- **So far:** we have seen how the flow of control changes as a *single program* executes
- **In reality:** multiple programs run *concurrently*
  - How does control flow across the many components of a computer system?
  - In particular, when there are more programs running than CPUs

- **Exceptional control flow** is a basic mechanism used for:
  - Transferring control between *processes* and *the operating system*
  - Handling *I/O* and *virtual memory* within the operating system
  - Implementing *multi-process* apps like shells and web servers
  - Implementing *concurrency*

Control Flow

- **Processors do only one thing:**
  - From start-up to shut-down, 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

```
<startup>
instr_1
instr_2
instr_3
…
instr_n
<shutdown>
```

Altering the Control Flow

- **Up to now: two ways to change control flow**
  - *Jumps* (conditional and unconditional)
  - *Call* and *Return*
  - Both react to changes in *program state*

- **Processor also needs to react to changes in *system state***
  - Unix/Linux user hits “Ctrl-C” at the keyboard
  - User clicks on a different application’s window on the screen
  - Data arrives from disk or a network adapter
  - Instruction divides by zero
  - System timer expires

- **Can jumps and procedure calls achieve this?**
  - No – the system needs mechanisms for “*exceptional*” control flow

Java Digression #1

- **Java has exceptions, but they're *something different***
  - Examples: NullPointerException, MyException, …
  - throw statements
  - try/catch statements

- **Java exceptions are for reacting to (unexpected) program state**
  - Can be implemented with stack operations and conditional jumps
  - A mechanism for "many call-stacks returns at once"
  - Requires additions to the calling conventions, but done with the features we covered

- **System-state changes on previous slide are mostly of different sort and are implemented differently!**

Exceptional Control Flow

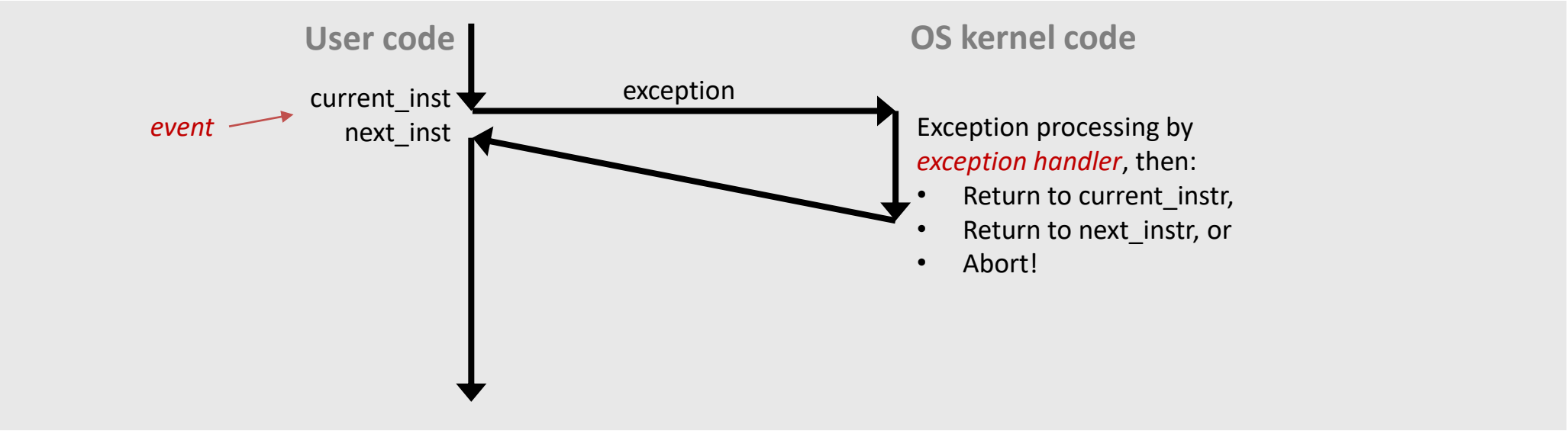
- **Exists at all levels of a computer system**

- **Low level mechanisms**
  - **Exceptions**
    - Change in processor's control flow in response to a system event
    - Implemented using a combination of hardware and OS software

- **Higher level mechanism**
  - **Process context switch**
    - Implemented by OS software and hardware timer
  - **Signals**
    - Implemented by OS software
    - We won't cover these today!

Exceptions

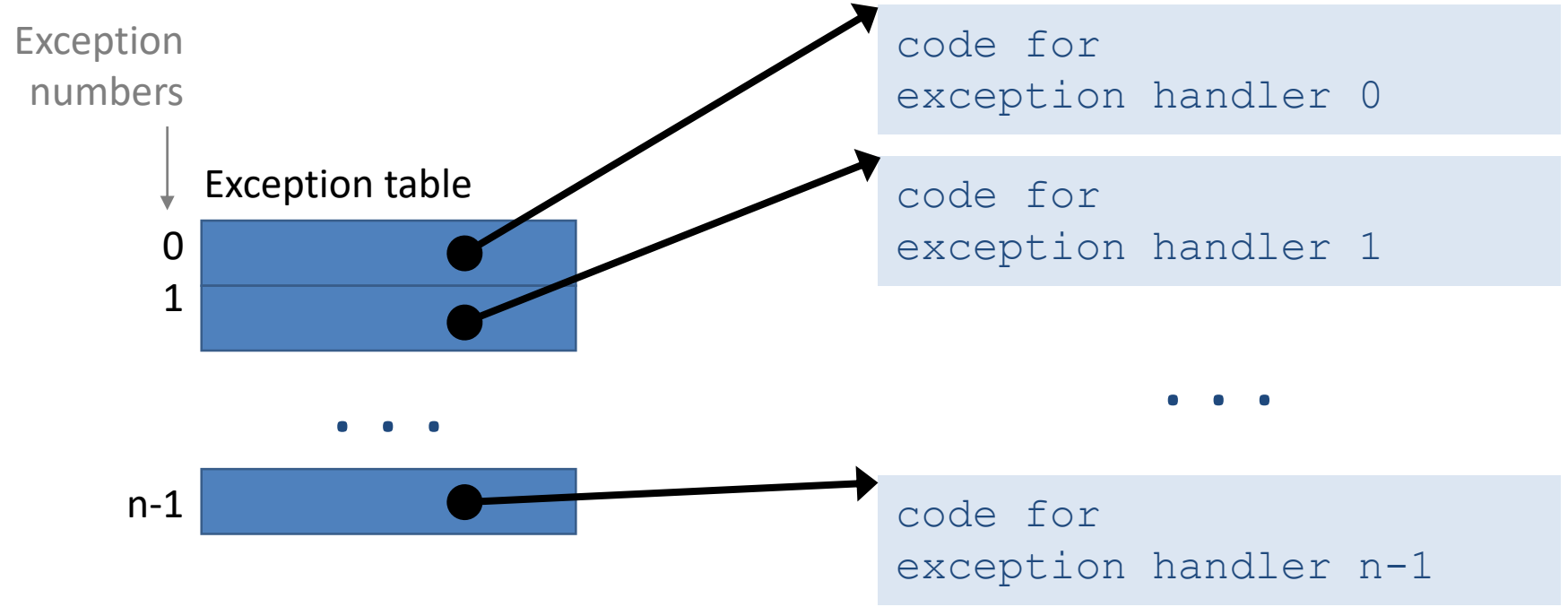
- **An *exception* is transfer of control to the operating system (OS) kernel in response to some *event*** (i.e., change in processor state)
  - Kernel is the memory-resident part of the operating system
  - Example events: division by 0, page fault, I/O request completes, Ctrl-C



- **How does the system know where to jump in the Operating System?**

Exception Table

- **A jump table for exceptions** (also called *Interrupt Vector Table*)
  - Each type of event has a unique exception number *k*
  - *k* = index into exception table (a.k.a interrupt vector)
  - *H*andler *k* is called each time the exception *k* occurs



Exception Table (Excerpt)

Exception Number	Description	Exception Class
0	Divide error	Fault
13	General protection fault	Fault
14	Page fault	Fault
18	Machine check	Abort
32-255	OS-defined	Interrupt or trap

For full list, check pp. 183:
http://download.intel.com/design/processor/manuals/253665.pdf

Today: System Control Flow

- Control Flow
- Exceptional Control Flow
- **Asynchronous exceptions (Interrupts)**
- **Synchronous exceptions (Traps and Faults)**

Asynchronous Exceptions (Interrupts)

- **Caused by events external to the processor**
  - Indicated by setting the processor's interrupt pin(s) (wire into CPU)
  - After interrupt handler runs, the handler returns to "next" instruction

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

Synchronous Exceptions

- **Caused by events that occur as a result of executing an instruction:**
  - ***Traps***
    - *Intentional*: transfer control to OS to perform some function
    - Example: ***system calls***, breakpoint traps, special instructions
    - Returns control to "next" instruction
  - ***Faults***
    - *Unintentional*: but possibly recoverable
    - Example: page faults (r), segment protection faults(u), integer divide-by-zero exceptions (u)
    - Either re-executes faulting ("current") instruction or aborts
  - ***Aborts***
    - *Unintentional*: and unrecoverable
    - Examples: parity error, machine check (hardware failure detected)
    - Aborts current program

System Calls

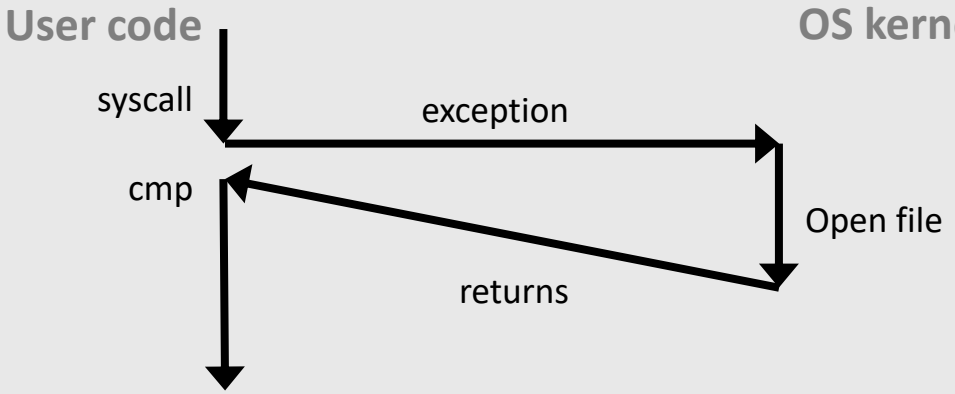
- Each system call has a unique ID number
- Examples for Linux on x86-64:

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 a process
62	`kill`	Send signal to process

Traps Example: Opening File

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

```
00000000000e5d70 <__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
```



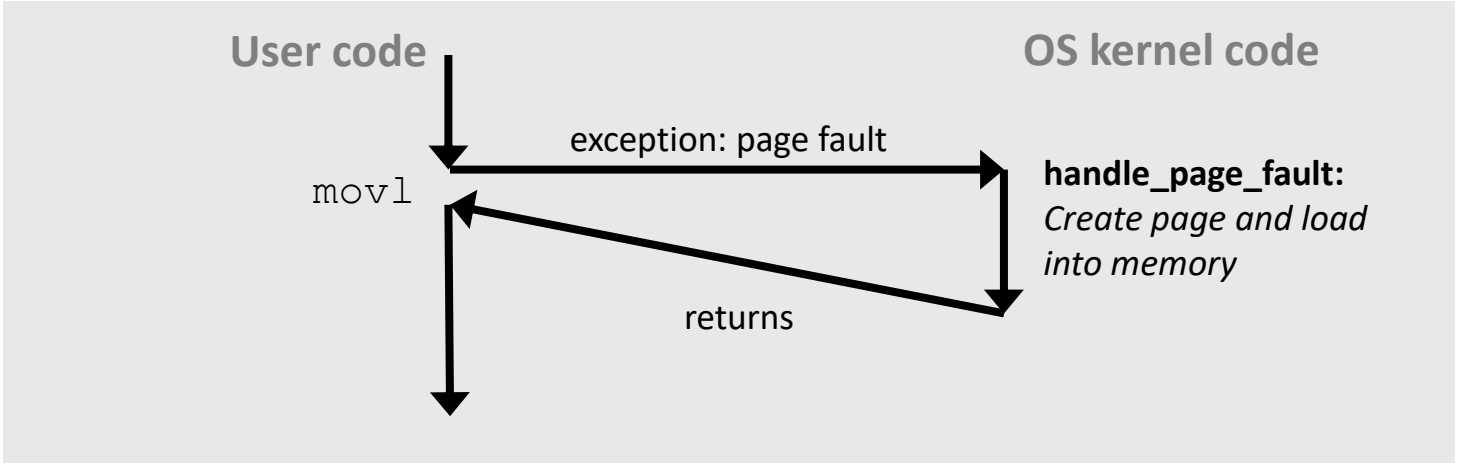
- `%rax` contains syscall number
- Other arguments in `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9`
- Return value in `%rax`
- Negative value is an error corresponding to negative `errno`

Fault Example: Page Fault

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

- 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
```

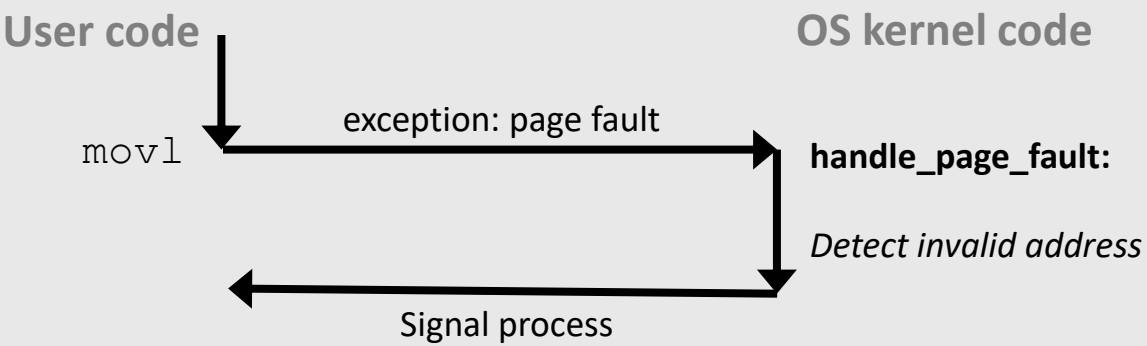


- Page fault handler must load page into physical memory
- Returns to faulting instruction: `movl` is executed again!
  - Successful on second try!

Fault Example: Invalid Memory Reference

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

```
80483b7:      c7 05 10 9d 04 08 0d          movl   $0xd, 0x804e360
```



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

Summary

- **Exceptions**
  - Events that require non-standard control flow
  - Generated externally (interrupts) or internally (traps and faults)
  - After an exception is handled, one of three things may happen:
    - Re-execute the current instruction
    - Resume execution with the next instruction
    - Abort the process that caused the exception