Systems Programming

Exceptions

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://lifeasageek.github.io

Lecture slides are prepared based on materials provided by CSAPP authors.

Today

- **Exceptional Control Flow** CSAPP 8
- Exceptions CSAPP 8.1

Control Flow

- **CPU/Processors do only one thing:**
  - Each CPU core simply reads and executes a sequence of instructions, one at a time *
  - This sequence is the CPU's *control flow*

CPU's control flow

Time

<startup>
inst_1
inst_2
inst_3
…
inst_n
<shutdown>

* Externally, from an architectural viewpoint (internally, the CPU may use parallel out-of-order execution)

Altering the Control Flow

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

- **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

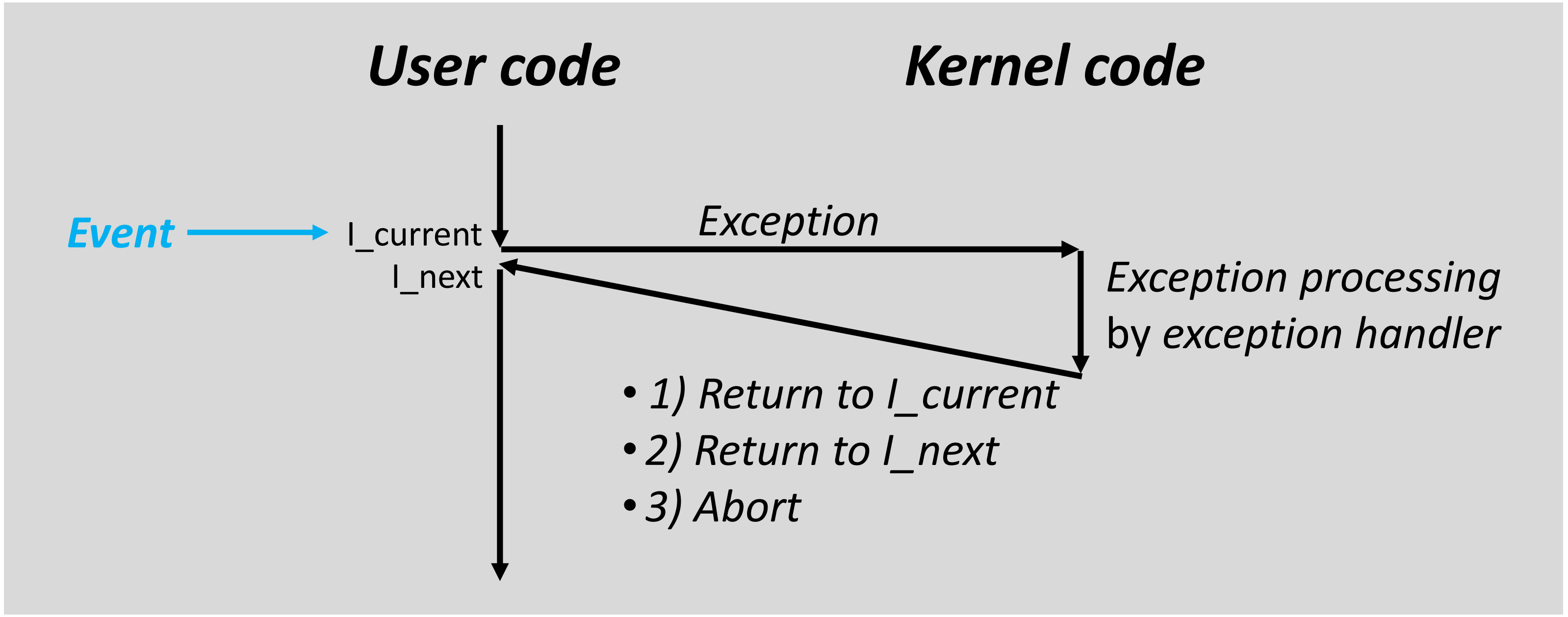
- **System needs mechanisms for "exceptional control flow"**

Today

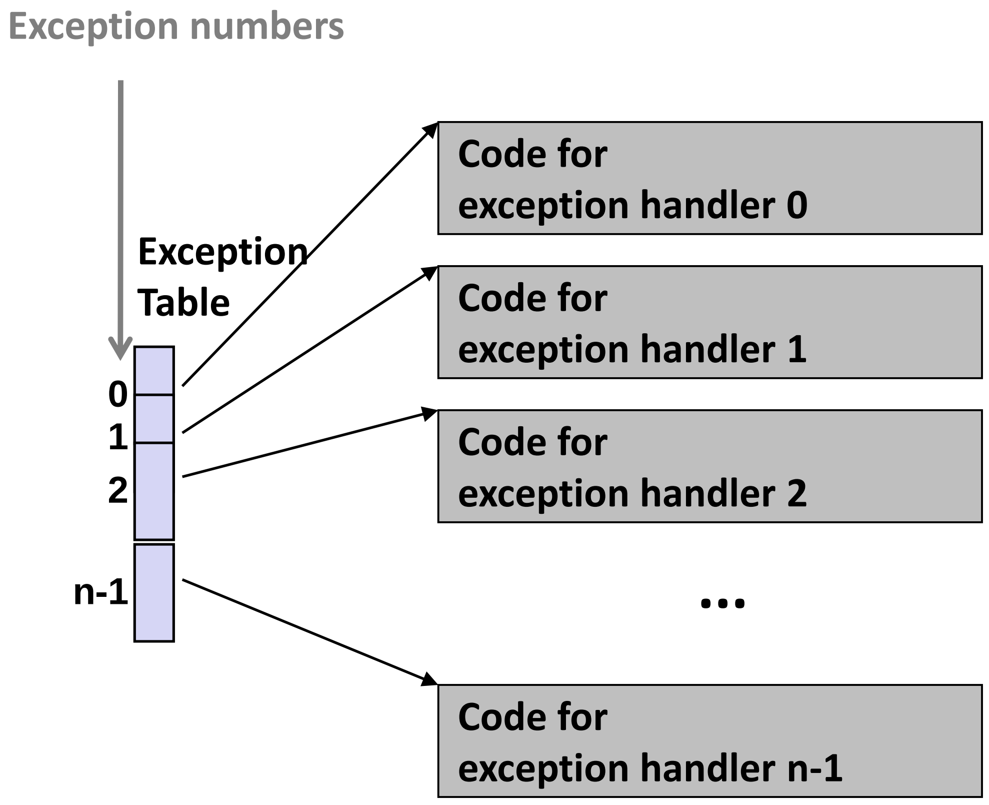
- Exceptional Control Flow
- **Exceptions**

Exceptions

- **An *exception* is a transfer of control to the OS *kernel* in response to some *event***
  - OS == Kernel == Privileged mode == Ring 0
  - Application == User == Unprivileged mode == Ring 3
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



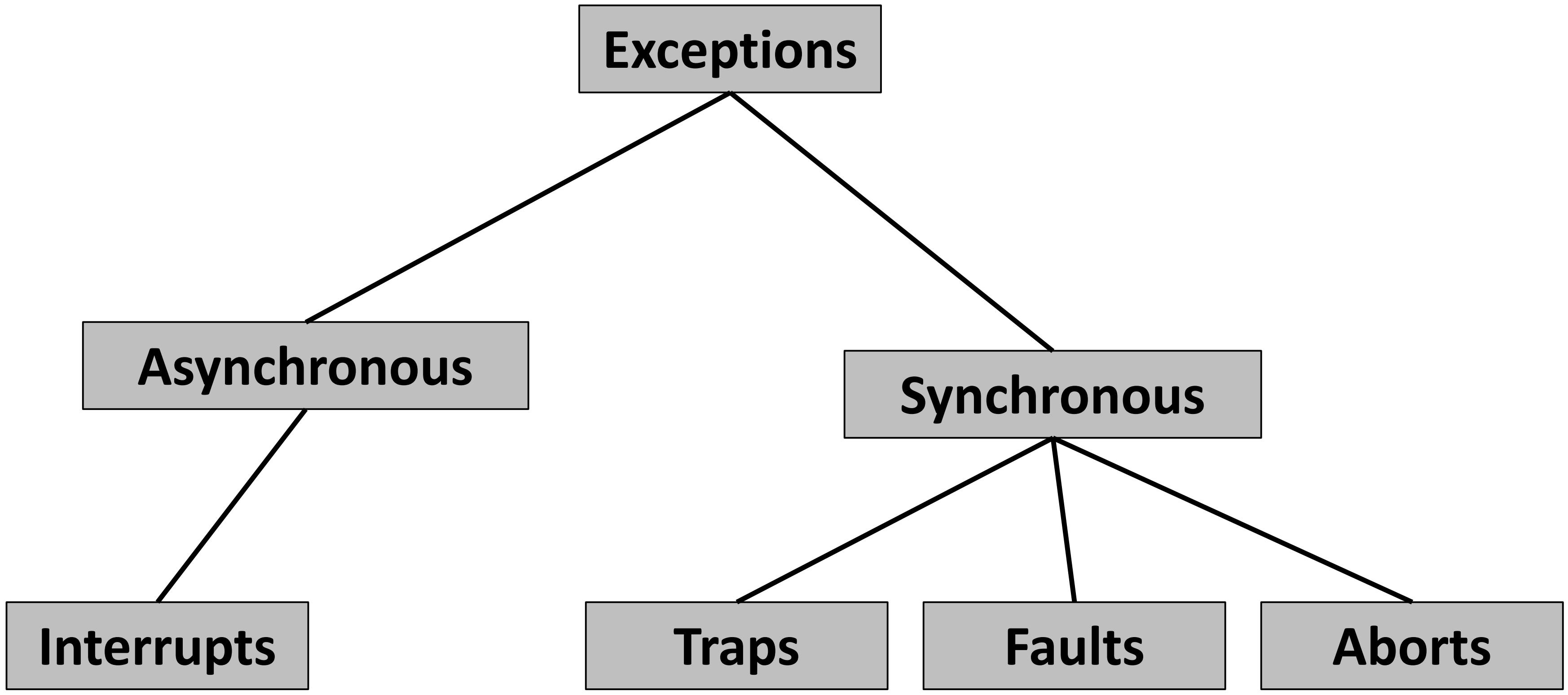
Exception Tables


Exception numbers
Exception Table
0
1
2
n-1
Code for exception handler 0
Code for exception handler 1
Code for exception handler 2
...
Code for exception handler n-1

- 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

- OS implements all code for exception handlers.
- OS prepares “Exception Table”, and let CPU know where “Exception Table” is.
- Upon receiving an event, CPU dispatches the exception to the corresponding exception handler.

(partial) Taxonomy



Asynchronous Exceptions (Interrupts)

- **Caused by events external to the processor**
  - External devices set the processor's *interrupt pin*
  - Kernel's 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
      - Called "kernel preemption"
  - **I/O interrupt** from external device
    - From keyboard: Hitting Ctrl-C at the keyboard
    - From NIC: Arrival of a packet from a network
    - From disk: Arrival of data from a disk

Synchronous Exceptions

- **Caused by events that occur as a result of executing an instruction:**
  - **Traps**
    - Intentional to ask for a certain pre-defined service
    - Examples: ***system calls***, gdb breakpoints
    - 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

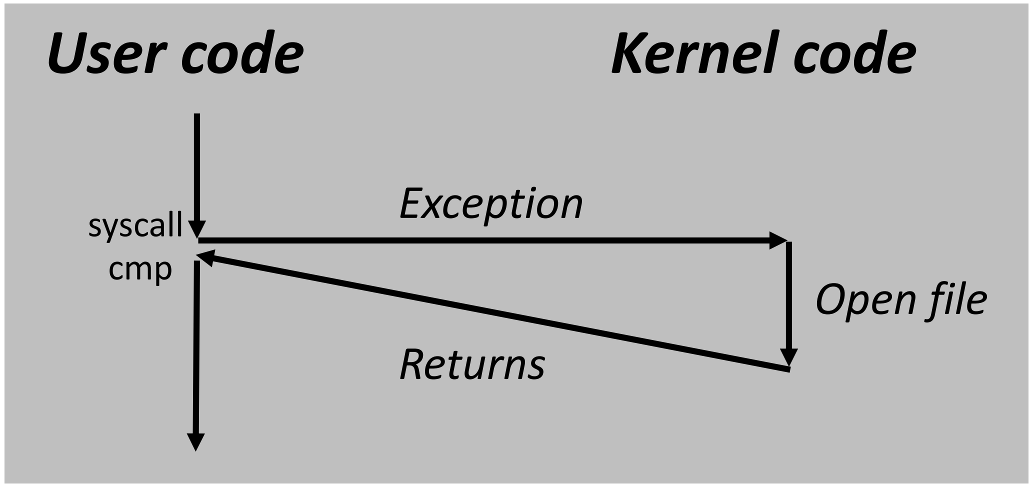
- **Each x86-64 system call in Linux has a unique ID number**
- **Examples:**

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`**

```
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 `%rax` is an error

System Call I

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

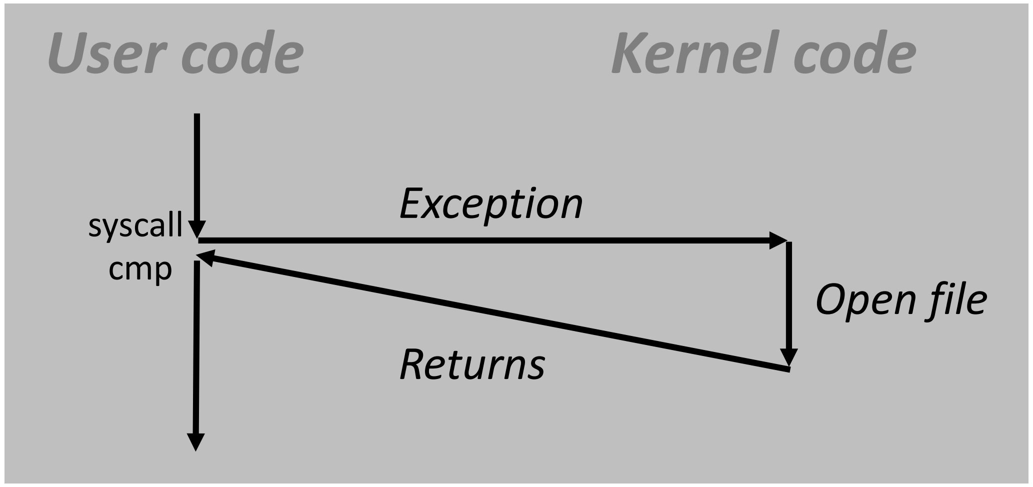
```
00000000000e5d70 <__
...
e5d79:   b8 02 00 00 00
e5d7e:   0f 05
e5d80:   48 3d 01 f0 ff ff
...
e5dfa:   c3                re
```

System call is almost like a function call

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

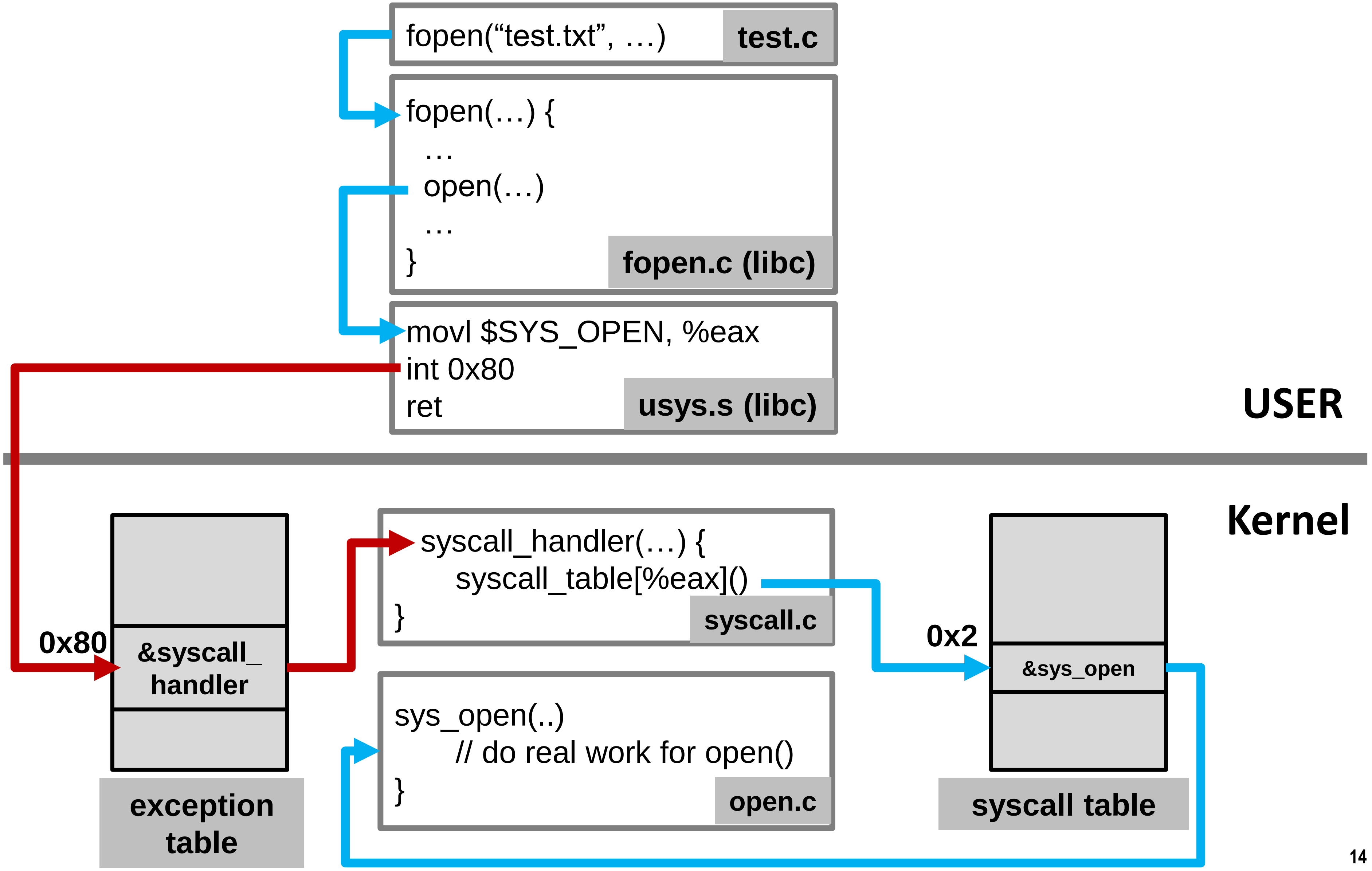
One Important exception!

- **Executed by Kernel**
- **Different set of privileges**
- **“index” of “function” is in `%rax`**



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

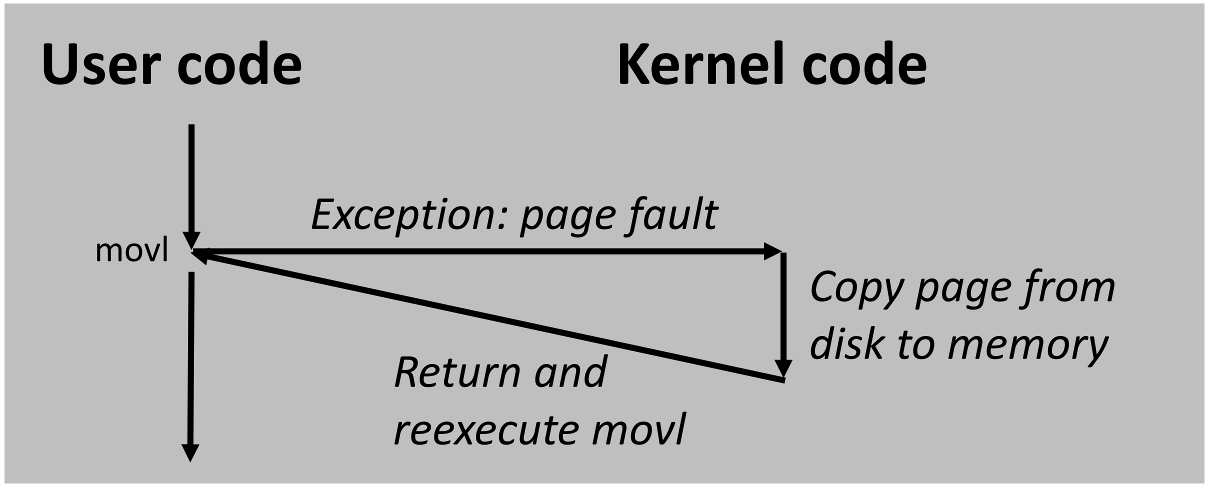
System Call Example: Opening File



Fault Example: Page Fault

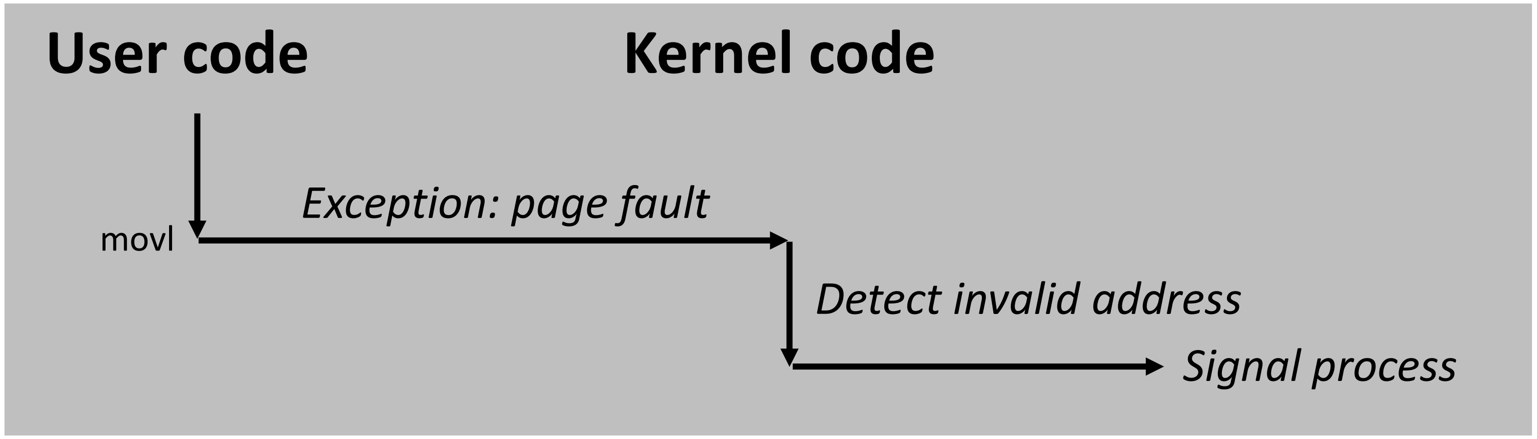
- User writes to memory location
- That portion (page) of user's memory is currently on disk

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



Fault Example: Invalid Memory Reference

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



- Kernel sends **`SIGSEGV`** signal to user process (will be covered later)
- User process exits with “segmentation fault”

Summary

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