Reading Reference: Textbook 1 Chapter 2

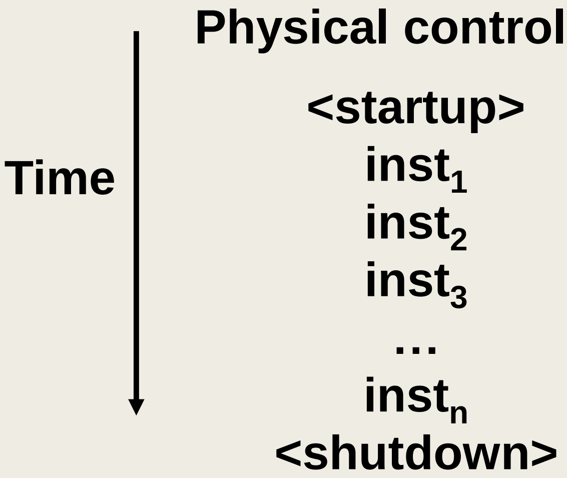
EXCEPTIONS CONTROL FLOW

TANZIR AHMED

CSCE 313, Spring 2020

Today’s Discussion: Control Flow

- Computers 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 system’s physical control flow (or flow of control)*



Physical control flow

Time

<startup>

inst$_1$

inst$_2$

inst$_3$

…

inst$_n$

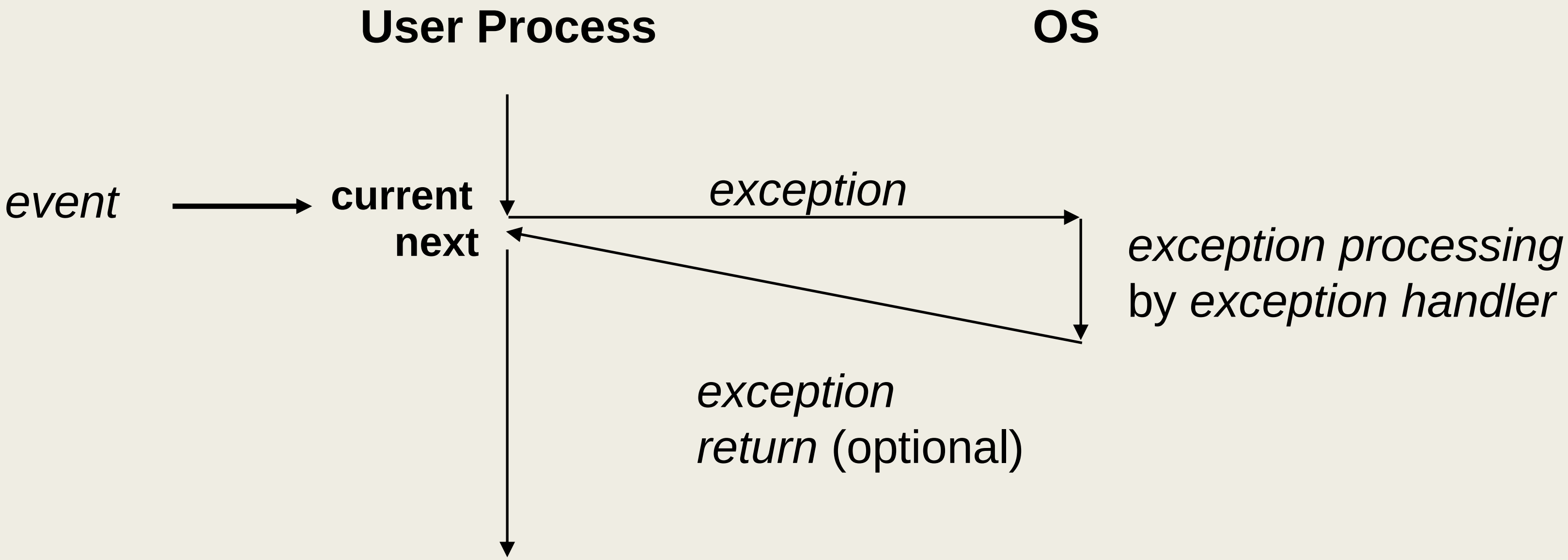
<shutdown>

Altering the Control Flow

- Program-assisted mechanisms for changing control flow:
  - ***Jumps and branches***—*react to changes in program state*
  - ***Function call and return*** *using stack discipline—react to program state*
- Insufficient for a useful system
  - *The user application is the central thing – how to let OS into the CPU unless the app gives up control?*
  - *Thus, difficult for the CPU to react to other changes in system state*
    - Data arrives from a network adapter
    - Instruction divides by zero
    - User hits control-C at the keyboard
- System needs mechanisms for “exception control flow”

Exception Control Flow

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

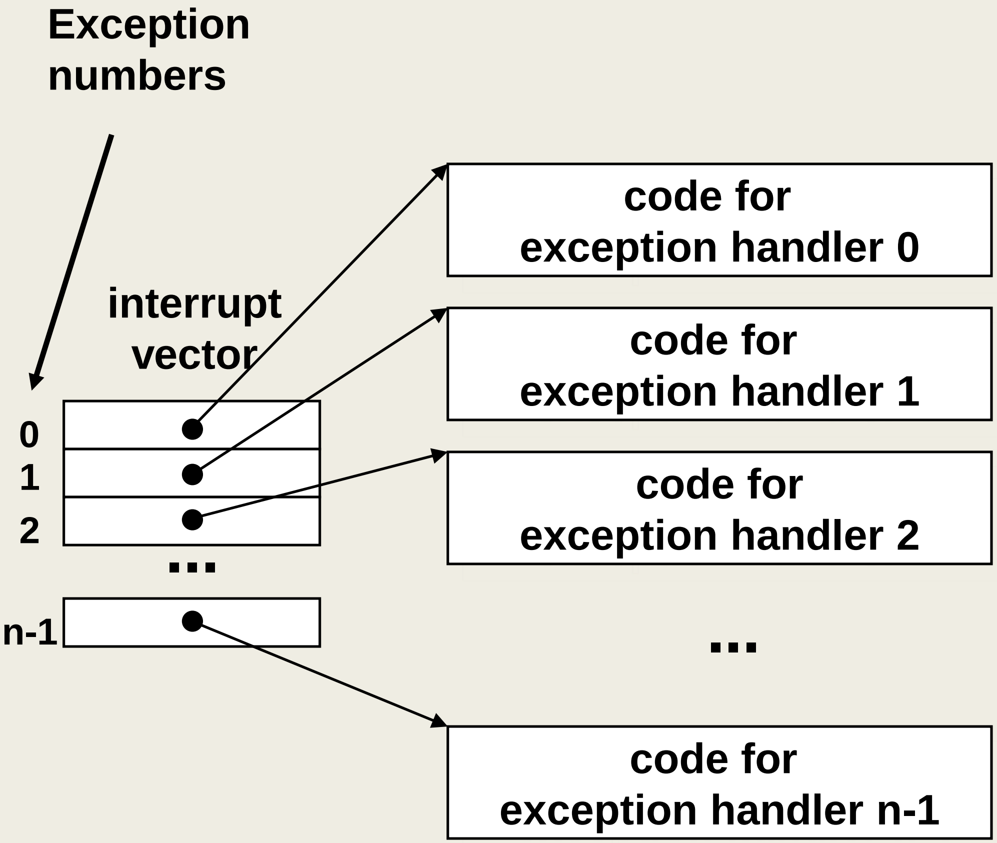


User Process
OS
event
current
next
exception
exception processing by *exception handler*
exception *return* (optional)

Asynchronous Exceptions (Interrupts)

- Caused by events **external** to processor (i.e., outside the current program)
  - *Indicated by setting the processor’s interrupt pin(s)*
  - *Handler returns to “next” instruction after servicing*
- **Examples:**
  - *I/O interrupts*
    - Key pressed on the keyboard
    - Arrival of packet from network, or disk
  - *Hard-reset interrupt*
    - Hitting reset button
  - *Soft-reset interrupt*
    - Hitting control-alt-delete to initiate restart on a PC

Interrupt Vectors

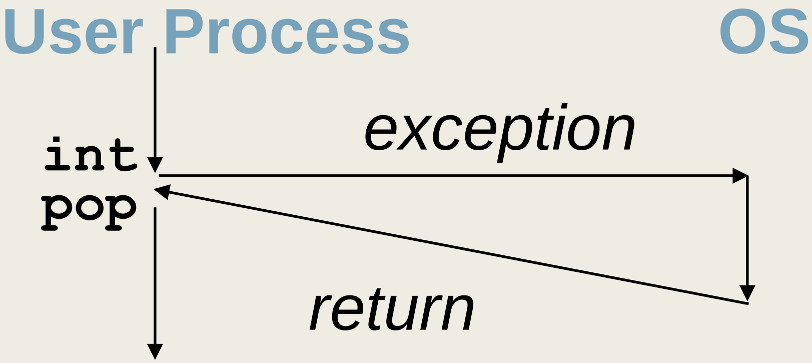

Exception numbers
interrupt vector
0
1
2
...
n-1
code for exception handler 0
code for exception handler 1
code for exception handler 2
...
code for exception handler n-1

- *Each type of event has a unique exception number k*
- *Index into jump table (a.k.a., interrupt vector)*
- *Jump table entry k points to a function (exception handler).*
- *Handler k is called each time exception k occurs.*

Synchronous Exceptions: Traps, Faults, Aborts

- Caused by events that occur as result of executing an instruction (i.e., from the currently running process):
- 3 types:
  - *Traps*
  - *Faults*
  - *Aborts*

Traps

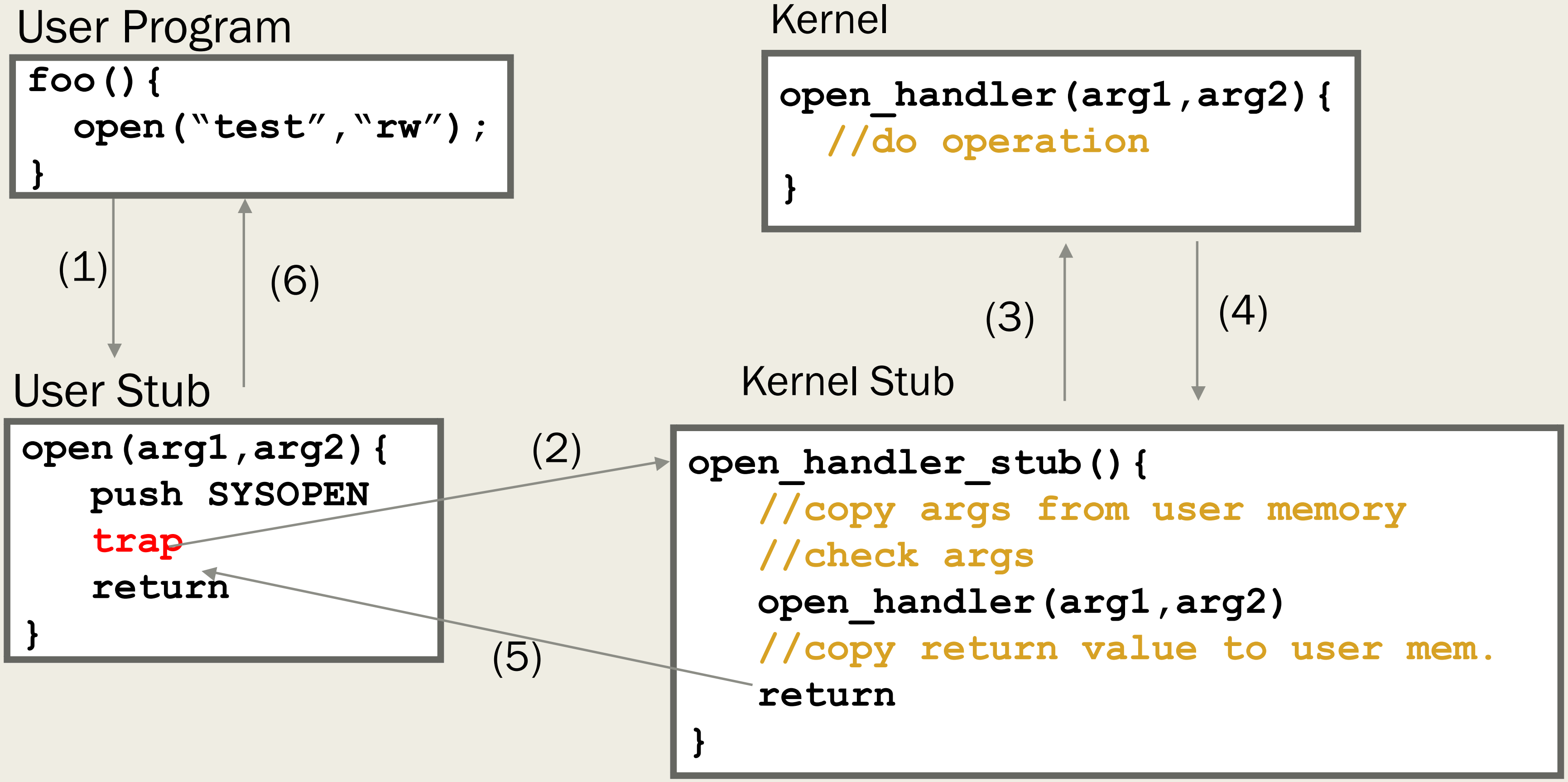


- Attributes
  - *Intentional*
  - *Returns control to "next" instruction*
  - *Examples: all **system calls** (e.g., printf, cout), breakpoint traps, special instructions*
- Example: Opening a File
  - *User calls* `open(filename, options)`
    - Function `open` executes system-call instruction: `int $0x80`
  - *OS must find or create file, get it ready for reading or writing*

Open file

  - *Returns integer file descriptor*

Flow of Control in System Calls



User Program

```
foo(){
   open(“test”,“rw”);
}
```

User Stub

```
open(arg1,arg2){
   push SYSOPEN
   trap
   return
}
```

Kernel

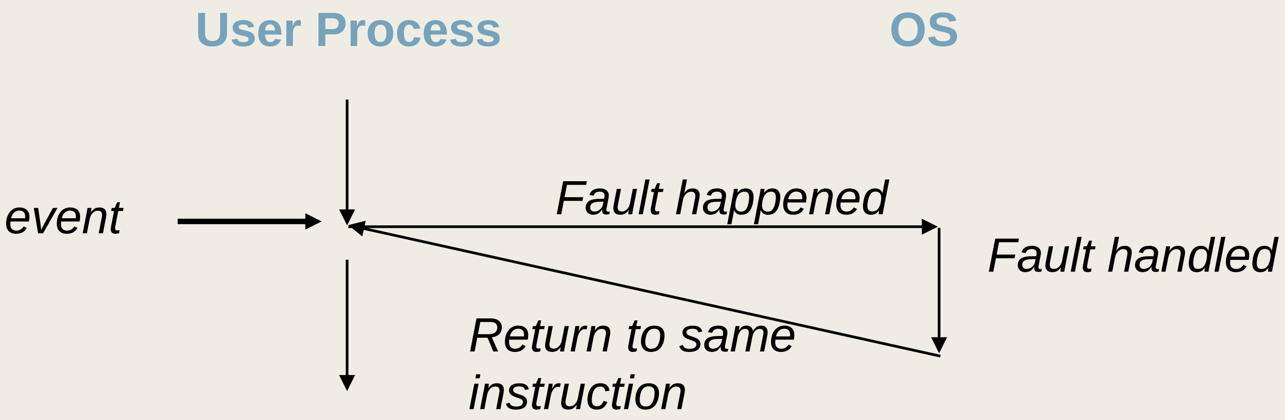
```
open_handler(arg1,arg2){
   //do operation
}
```

Kernel Stub

```
open_handler_stub(){
   //copy args from user memory
   //check args
   open_handler(arg1,arg2)
   //copy return value to user mem.
   return
}
```

Faults

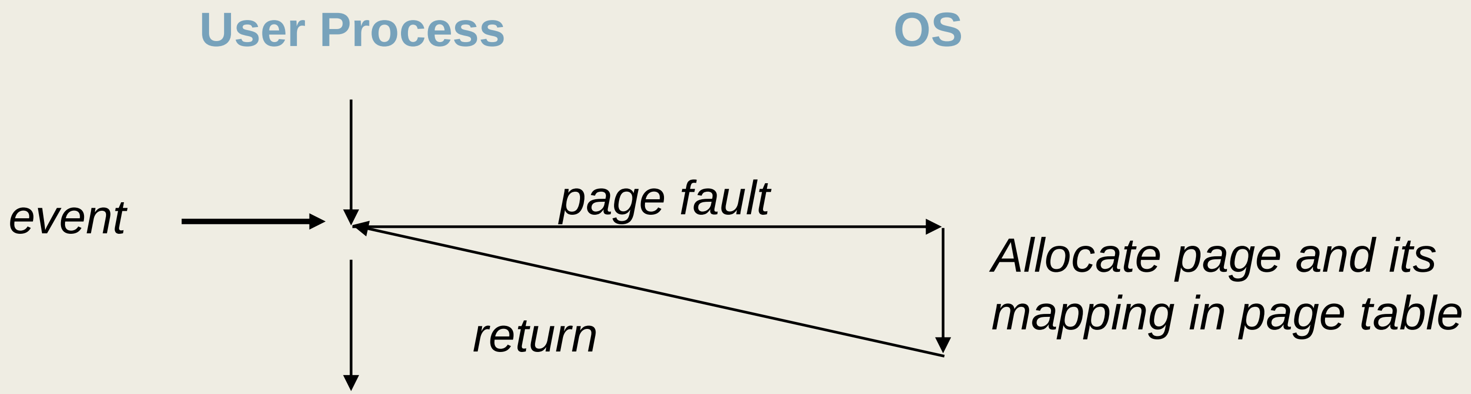
- Attributes
  - *Unintentional but possibly recoverable*
  - *Examples: Page Faults*
  - *Either re-executes faulting (“current”) instruction or aborts*



Fault Example #1

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

- **Memory Reference**
  - *User writes to memory location*
  - *That portion (page) of user's memory is not mapped yet (because memory pages are mapped only when necessary)*
  - *Page handler must load page into physical memory*
  - *Returns to faulting instruction*
  - *Successful on second try*



User Process

OS

event

page fault

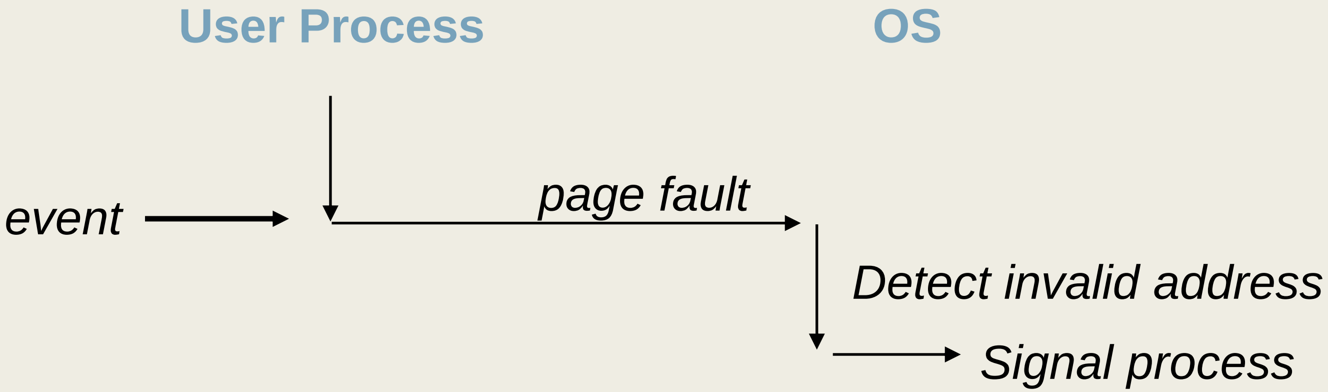
return

Allocate page and its mapping in page table

Fault Example #2

- **Illegal Memory Reference**
  - *User writes to memory location*
  - *Address is not valid*
  - *Page handler detects invalid address*
  - Sends *SIGSEGV* signal to user process
  - *User process exits with "segmentation fault"*

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



User Process

OS

event

page fault

Detect invalid address

Signal process

Aborts

- Attributes
  - *Unintentional and unrecoverable*
  - *Examples: parity error, machine check, divide by zero*
  - *Aborts current program or entire OS*

Summarizing Control Flow Exceptions

- User programs are not in charge of (and therefore not burdened with) handling everything that the OS does not like
  - *If you divided by 0, it was probably a mistake anyways*
- Mechanism is used by OS to do things beyond error handling
  - *E.g., page faults are used to enable "lazy" physical memory allocation*
- Are Synchronous/Internal (Traps, Faults, Aborts) OR Asynchronous/External (I/O Interrupts, Hard or Soft Reset etc.)

In Closing

- *Today we learnt the importance and various forms of how applications eventually get the attention of underlying System Hardware and Software (privileged code to keep sanity, illusions, and glues)*
- *We saw an interesting analogy with a coffee shop and also looked at some real CS System examples illustrating exception control flow.*
- *Next, we will dive a little deeper into Dual Mode Operation. Read Chapter 2 in entirety to prepare for an interesting discussion.*