CS 354 - Machine Organization & Programming
Thursday, November 21, 2019

Project p5 (4.5%): DUE at 10 pm on Monday, December 2nd
Project p6 (4.5%): Assigned on Tuesday, November 26th
Homework hw7 (1.5%): DUE at 10 pm Wednesday, November 27th

Last Time

Unions
Pointers
Function Pointers
Buffer Overflow & Stack Smashing
Flow of Execution
Exceptional Events
Kinds of Exceptions

Today

Kinds of Exceptions (from last time)
Transferring Control via Exception Table
Exceptions in IA-32 & Linux
Processes and Context
User/Kernel Modes
Context Switch
Context Switch Example

Next Time

Signals
Read: B&O 8.5 intro, 8.5.1 - 8.5.3, 8.5.4 p. 745

Transferring Control via Exception Table

Transferring Control to an Exception Handler

1. push return addr (I_{curr} or I_{next})

2. push interrupted process's state so it can be restarted.
   context switch.

→ What stack is used for the push steps above?
kernel's stack if it takes construc
otherwise it's the interrupted proc's stack.

3. do indirect function call, which runs the appropriate excution handware.

indirect function call M[R[ETBR] + Enum]
exection table register.

EHA is for exception handler's address

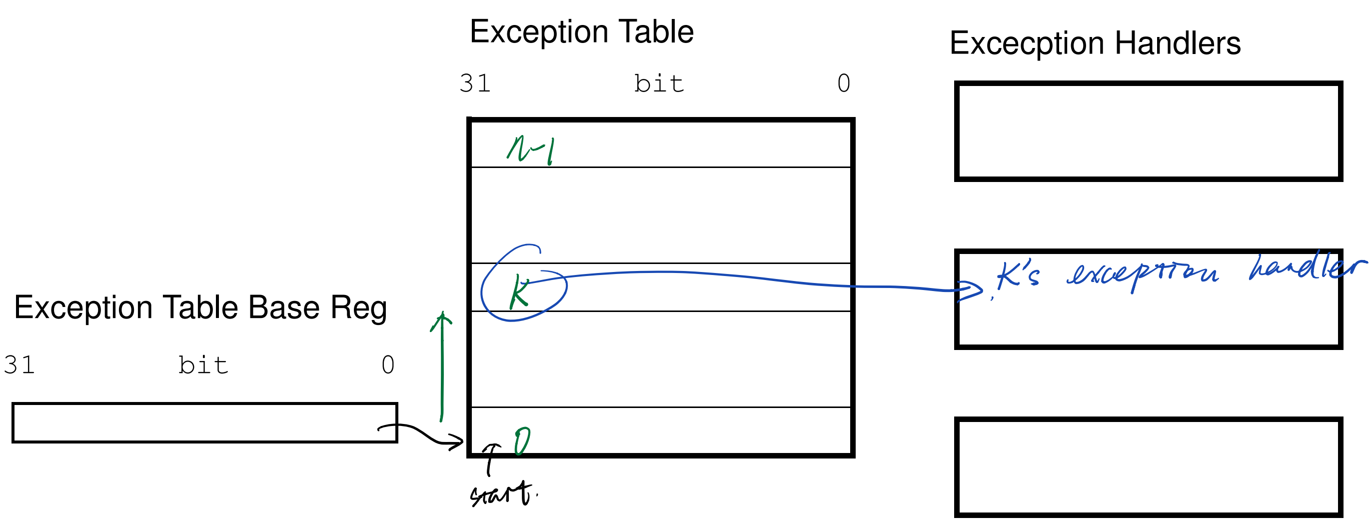
ETBR is for exception table base reg

ENUM is for exception number

Exception Table

is a jump table for exceptions that's allocated by OS on system boot

exception number (Enum)
unique none-negative integer associate with each exception type.



Exceptions in IA-32 & Linux

Exception Numbers and Types

0 - 31 are defined by processor

0 divide by 0
13 protection fault. ⇒ seg fault.
14 page fault.
18 machine check (hardware)

32 - 255 are defined by OS

128 ($0x80) trap (make a system call)

System Calls and Service Numbers

1 exit terminate the process
2 fork create a new process from existing process
3 read file 4 write file 5 open file 6 close file
11 execve starts a new program.

Making System Calls

1.) put service number in %eax

2.) put sys. call args in remaining regs except for %esp

3.) `int $0x80` 128_{10}

System Call Example

```
#include <stdlib.h>
int main(void) {
    write(1, "hello world\n", 12);
    exit(0);
}
```

Assembly Code:

```
.section .data
string:
    .ascii "hello world\n"
string_end:
    .equ len, string_end - string
.section .text
.global main
main:
    movl $4, %eax        - 1
    movl $1, %ebx        2
    movl $string, %ecx
    movl $len, %edx
    int $0x80            3
    movl $1, %eax        - 1
    movl $0, %ebx        - 2
    int $0x80
```

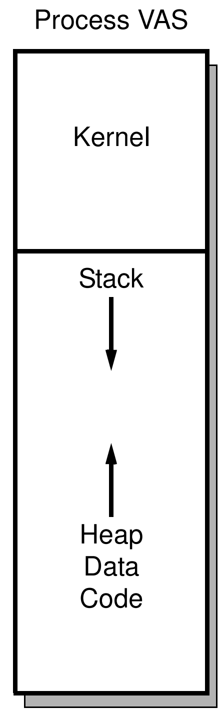
Copyright © 2016-2019 Jim Skrentny

int $0x80 Processes & Context

Recall, a <u>process</u>

- is an instance of a running program
- has context, which is all the info needed to restart it.

Why? easier to treat a process as a single entity running by itself.



Key illusions process exclusively uses

1. CPU
2. MEM
3. Devices

→ Who is the illusionist? OS

Concurrency

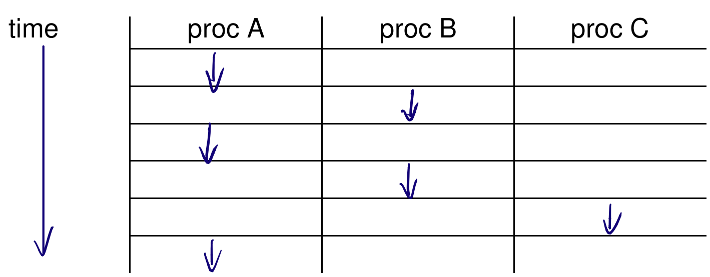
combined execution of 2 or more processes

<u>scheduler</u> kernel code to switch among processes

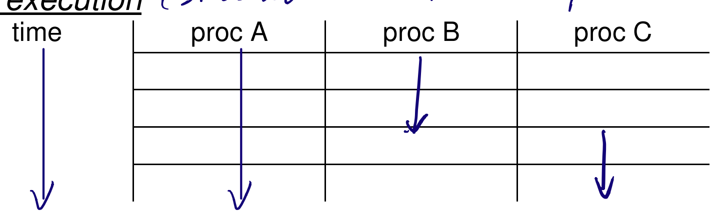
<u>interleaved execution</u>

one CPU is shared among processes.

<u>time slice</u> (AKA quantnum) interval of time of a proc gets CPU



<u>parallel execution</u> (simoutaneous) multiple CPU so each process gets its own



core ~~0~~ core 1 Core 2

User/Kernel Modes

What? Processor <u>*modes*</u> are different primitive levels that a process can run in

<u>*mode bit*</u> indicates execution mode

kernel mode (1) can execute any instruction
can access any mem
can access any device

user mode (0) exec of some instruction is restricted
access to mem is limited
device access is through OS

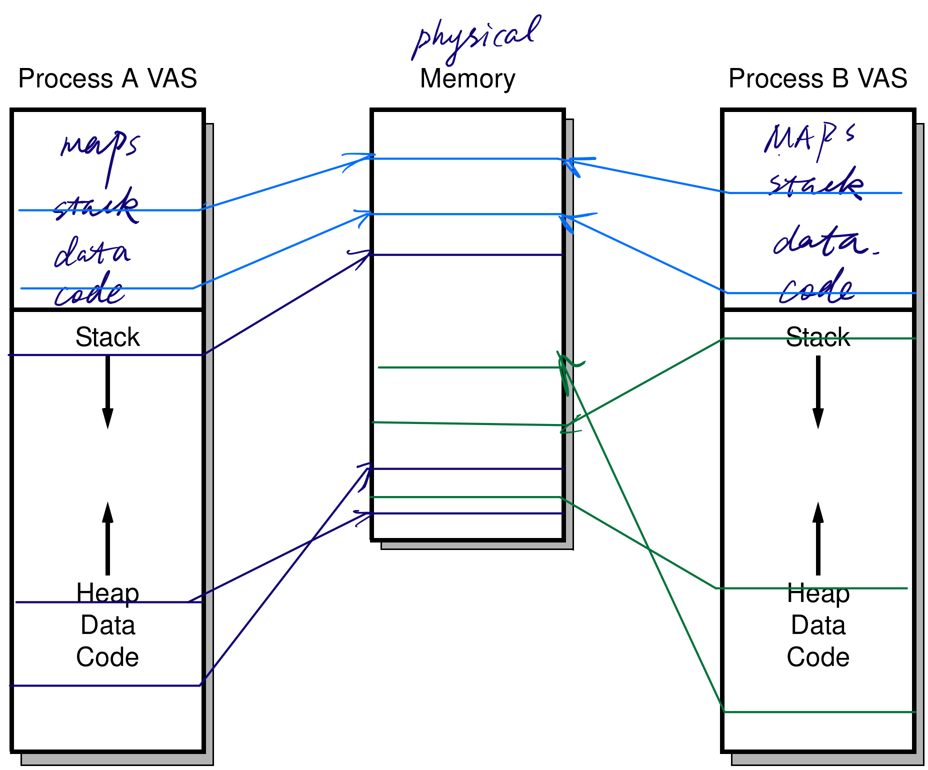
flipping modes

- start in user mode
- only an exception can switch to kernel mode
- kernel handler can switch back to user mode.

Sharing the Kernel

mem resident part of OS

shared among all user processes



Context Switch

What? A <u>context switch</u>
- is when the OS switches out one running program for another
- requires presenvation of processes's context
  1. CPU state
  2. user stack
  3. kernal's stack.
  4. kernel's data structure a. page table.
     b. process table.
     c. file table.

When? start of kernel execution
as a new process begins

Why? it enables exceptions to processed

How?

1. save context of current proc
2. restore context of new or restored proc
3. transfer control to new or restored proc.

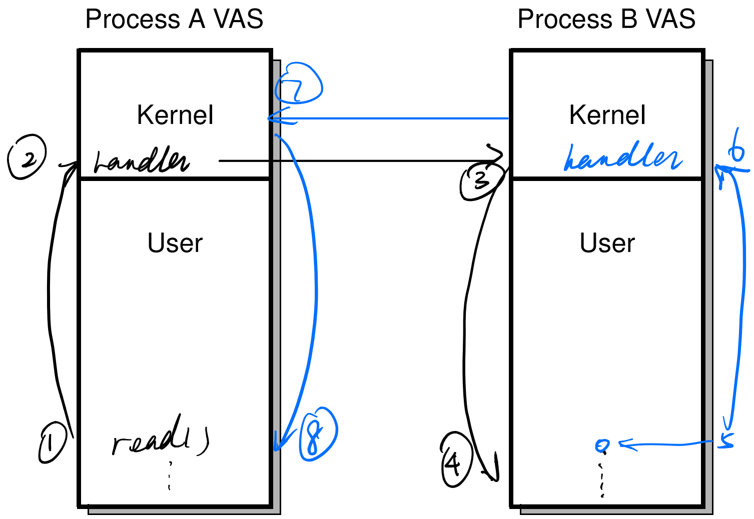
❋ *Context switches* are very expensive.

→ What is the impact of a context switch on the cache?

negative

Context Switch Example

read()



1. in user mode, proc A running -- read()
   service #3, trap #128 (int x080)

2. switch to kernel mode, run handler.
   changes DMA (Direct Memory Access) so
   Drive controler writes disk pages directly to mem

3. in kernel mode, do context switch.
   perserve proc. A's context, restore proc B's context.
   why? reading from 2nd storage is very slow, so use CPU for some other processes.

4. switch to user mode, run proc A.

5. in user mode, proc B is running
   interrupt from drive controler DMA is done.

6. switch to kernel mode, run handler

7. in kernel mode do context switch
   preserve B's context, restore A's context.

8. switch to user mode, to resume proc A.
   execute after read()