CS 354 - Machine Organization & Programming
Tuesday Nov 26, 2024

TA Consulting, Peer Mentoring end at 4pm on Wednesday and resume Monday Nov 30th.
Happy Thanksgiving!

Homework hw7: DUE on or before Monday Dec 2

Homework hw8: DUE on or before Monday Dec 5

Project p5 : DUE on or before Wednesday Nov 27

Project p6: Available and due on last day of classes.

Learning Objectives

- Describe and explain how computers transfer control to other processes
- Diagram and describe Exception Table and its use.
- Identify by name, number, and use several common exception types.
- Identify by name, number, and use several common system call operations.
- Describe and trace assembly for system calls.
- Describe and explain a process'es context.
- Diagram and describe interleaved processes and parallel processes
- Describe and explain the role of the Kernel's scheduler.
- Compare and constrast kernel mode vs user mode.
- Identify and describe the steps and state changes in a context switch.

This Week

<table>
<tr><td>Finish Week 12 outline
Transferring Control via Exception Table
Exceptions/System Calls in IA-32 & Linux
Processes and Context
User/Kernel Modes
Context Switch
Context Switch Example</td><td><b>Next Week</b>
Meet Signals
Three Phases of Signaling
Processes IDs and Groups
Sending Signals
Receiving Signals</td></tr>
<tr><td colspan="2"><b>This Week and Next Week</b>: Signals, and multifile coding, Linking and Symbols
B&O 8.5 Signals Intro, 8.5.1 Signal Terminology
8.5.2 Sending Signals
8.5.3 Receiving Signals
8.5.4 Signal Handling Issues, p.745</td></tr>
</table>

Transferring Control via Exception Table

✵ *Exceptions transfer control*

Transferring Control to an Exception Handler

1. push

2. push

→ What stack is used for the push steps above?

3. do indirect function call

indirect function call

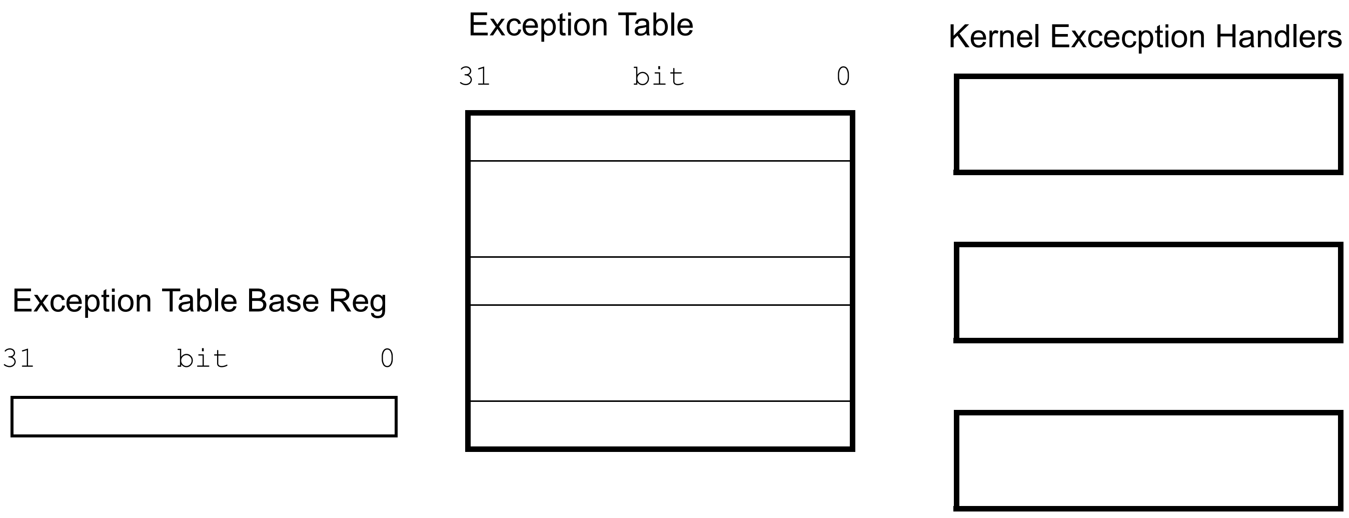
ETBR is for exception table base reg

ENUM is for exception number

EHA is for exception handler's address

Exception Table

exception number



Copyright © 2016-2024 Jim Skrentny

Exceptions/System Calls in IA-32 & Linux

Exception Numbers and Types

0 - 31 are defined by processor 0
13
14
18

32 - 255 are defined by OS 128 ($0x80)

System Calls and Service Numbers

1 exit
2 fork
3 read file 4 write file 5 open file 6 close file
11 execve

Making System Calls

1.)

2.)

3.) `int $0x80`

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
   movl $1, %ebx
   movl $string, %ecx
   movl $len, %edx
   int $0x80
   movl $1, %eax
   movl $0, %ebx
   int $0x80
```

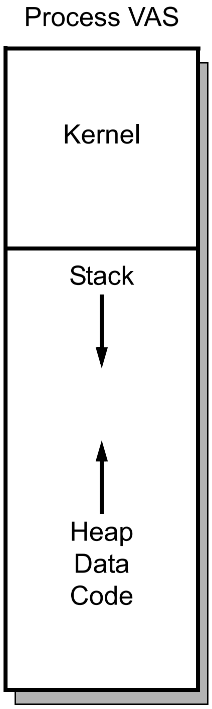
Processes & Context

Recall, a *process*

◆

◆

Why?



Key illusions

→ Who is the illusionist?

Concurrency

scheduler

interleaved execution

time slice

time	proc A	proc B	proc C

parallel execution

time	proc A	proc B	proc C

User/Kernel Modes

What? Processor *modes* are

mode bit

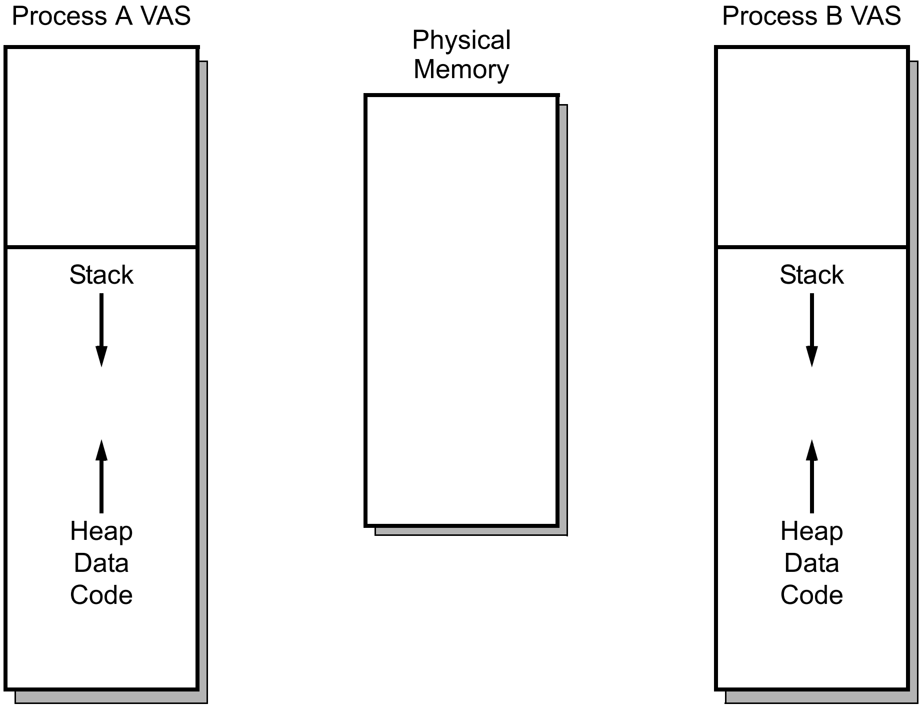
kernel mode

user mode

flipping modes

-
-
-

Sharing the Kernel



Context Switch

What? A *context switch*

-

-

When?

Why?

How?

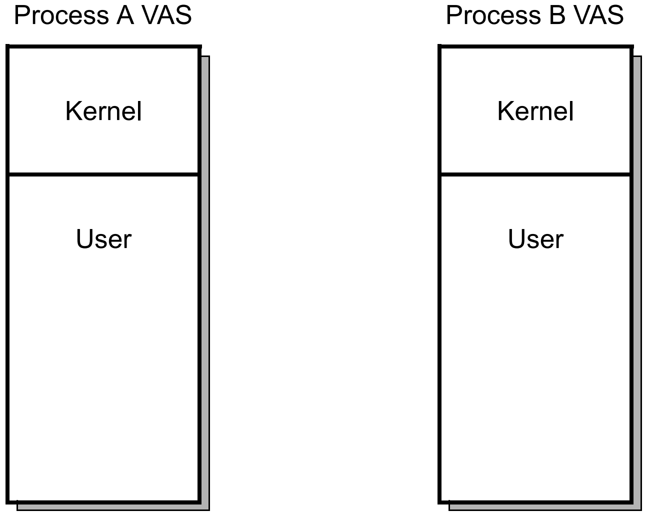
1.
2.
3.

✵ *Context switches*

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

Context Switch Example

Stepping through a `read()` System Call



1.

2.

3.

4.

5.

6.

7.

8.

Meet Signals

✵ *The Kernel uses signals*

What? A *signal* is

Linux:

```
$kill -l
```

signal(7)

Why?

◆

1.

2.

◆

◆

Examples

1. divide by zero

exception interrupts to kernel handler

- kernel signals user proc with

2. illegal memory reference

exception interrupts to kernel handler

- kernel signals user proc with

3. keyboard interrupt

- ctrl-c interrupts to kernel handler which

- ctrl-z interrupts to kernel handler which

Three Phases of Signaling

Sending

- when the kernel
-

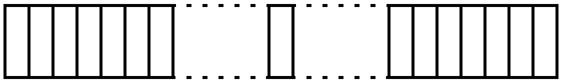
Delivering

when the kernel

pending signal

-

bit vectors



-

Receiving

when the kernel

-
-

blocking

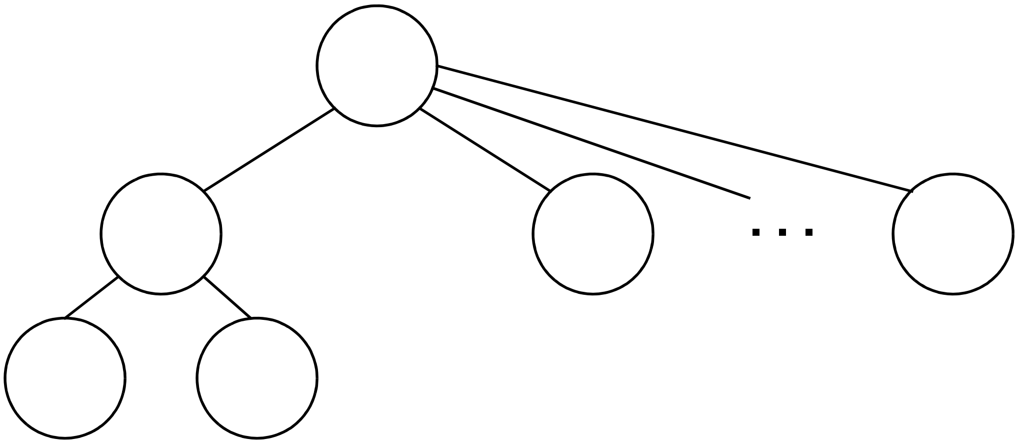
-
-

Process IDs and Groups

What? Each process

-

-



Why?

How?

Recall: ps

getpid(2)
getpgrp(2)

```
#include

pid_t getpid(void)

pid_t getpgrp(void)
```

Sending Signals

What? A signal is sent by the kernel or a user process via the kernel

How? Linux Command

kill(1)

```
kill -9  <pid>
```

→ What happens if you kill your shell?

How? System Calls

kill(2)

killpg(2)

```
#include <sys/types.h>
#include

   int kill (pid_t pid, int sig)
```

alarm(2)

```
#include

unsigned int alarm(unsigned int seconds)
```

Receiving Signals

What? A signal is received by its destination process

How? Default Actions

- Terminate the process
- Terminate the process and dump core
- Stop the process
- Continue the process if it’s currently stopped
- Ignore the signal

How? Signal Handler

1.
    -
    -
2.
    -

    ~~signal(2)~~
    sigaction(2)

Code Example

```
#include <signal.h>
#include ...
#include <string.h>

void handler_SIGALRM() { ... }

int main(...) {
```