CS 354 - Machine Organization & Programming
Tuesday April 18, and Thursday April 20, 2023

Homework hw6: DUE on or before Monday Apr 17

Homework hw7: DUE on or before Monday Apr 24

Project p5: DUE on or before Friday April 21st

Project p6: Assigned soon and Due on May 5th, last day of classes. No late day, no Oops on p6.

Last Week

Function Call-Return Example (L20 p7) Recursion Stack Allocated Arrays in C Stack Allocated Arrays in Assembly Stack Allocated Multidimensional Arrays	Stack Allocated Structs Alignment Alignment Practice Unions

This Week

Pointers Function Pointers Buffer Overflow & Stack Smashing Flow of Execution Exceptional Events Kinds of Exceptions	Transferring Control via Exception Table Exceptions/System Calls in IA-32 & Linux Processes and Context User/Kernel Modes Context Switch Context Switch Example
Next Week: 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	

Pointers

Recall Pointer Basics in C

```
int i = 11;
int *iptr = &i;
*iptr = 22;
```

pointer type `int *`

pointee type used by compiler to determine the scaling factor used in ASM

pointer value `0x2A300F87`, `0x00000000` (NULL)
address used with addressing modes to specify an effective address in ASM

address of `&i`
& operator, becomes `leal` instr, which just calculates the effective address

dereferencing `*iptr`
* operator, becomes `mov` instr, which accesses mem at the effective address

Recall Casting in C

```
int *p = malloc(sizeof(int) * 11);

... (char *)p + 2
```

- *Casting changes the scaling factor used not the pointer's value.*

Function Pointers

What? A *function pointer*

-
-

Why?

enables functions to be

-

-

How?

```
int func(int x) { ...}      //1. implement some function

int (*fptr)(int);           //2. declare function pointer

fptr = func;                //3. assign its function

int x = fptr(11);           //4. use function pointer
```

Example

```
#include <stdio.h>

void add     (int x, int y) { printf("%d + %d = %d\n", x, y, x+y); }
void subtract(int x, int y) { printf("%d - %d = %d\n", x, y, x-y); }
void multiply(int x, int y) { printf("%d * %d = %d\n", x, y, x*y); }

int main() {
    void (*fptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int choice;
    int i = 22, j = 11; //user should input

    printf("Enter: [0-add, 1-subtract, 2-multiply]\n");
    scanf("%d", &choice);
    if (choice > 2) return -1;
    fptr_arr[choice](i, j);
    return 0;
}
```

Bounds Checking

```
int a[5] = {1,2,3,4,5};
printf("%d", a[11]);
```

→ What happens when you execute the code?

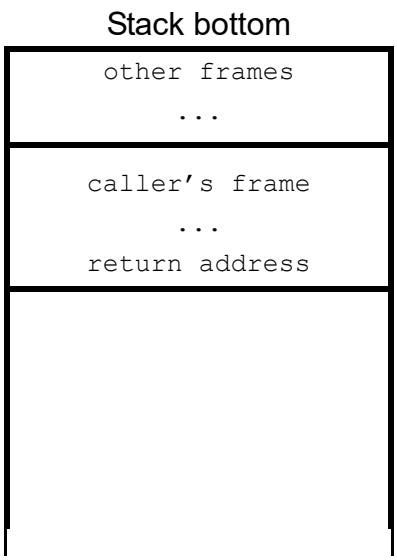
❋ *The lack of bounds checking array accesses is one of C's main vulnerabilities.*

Buffer Overflow

◆

◆

```
void echo() {
   char bufr[8];
   gets(bufr);
   puts(bufr);
}
```



❋ *Buffer overflow can overwrite data outside the buffer.*

❋ *It can also overwrite the state of execution!*

Stack Smashing

1. Get "exploit code" in
   enter input crafted to be machine instrs

2. Get "exploit code" to run
   overwrite return address with addr of buffer with exploit code

3. Cover your tracks
   restore stack so execution continues as expected

❋ *In 1988 the Morris Worm brought down the Internet using this kind of exploit.*

Flow of Execution

What?

control transfer

control flow

- What control structure results in a smooth flow of execution?

- What control structures result in abrupt changes in the flow of execution?

Exceptional Control Flow

logical control flow

exceptional control flow

event

processor state

Some Uses of Exceptions

process

OS

hardware

Exceptional Events

What? An *exception*

-
-
-

→ What's the difference between an asychronous vs. a synchronous exception?

asynchronous

synchronous

General Exceptional Control Flow

	Application	Exception Handler
0. normal flow	I_0	
	I_1	
1.		
2.		
3.		
4.		

Kinds of Exceptions

→ Which describes a ***Trap***? ***Abort***? ***Interrupt***? ***Fault***?

1.

signal from external device
asynchronous
returns to Inext

How? Generally:

1.

2.

3. transfer control to appropriate exception handler

4. transfer control back to interrupted process's next instruction

vs. polling

2.

intentional exception
synchronous
returns to Inext

How? Generally:

1.

`int`

2. transfer control to the OS system call handler

3. transfer control back to process's next instruction

3.

potentially recoverable error
synchronous
might return to Icurr and re-execute it

4.

nonrecoverable fatal errors
synchronous
doesn't return

Transferring Control via Exception Table

❋ *Exceptions transfer control to the Kernel.*

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

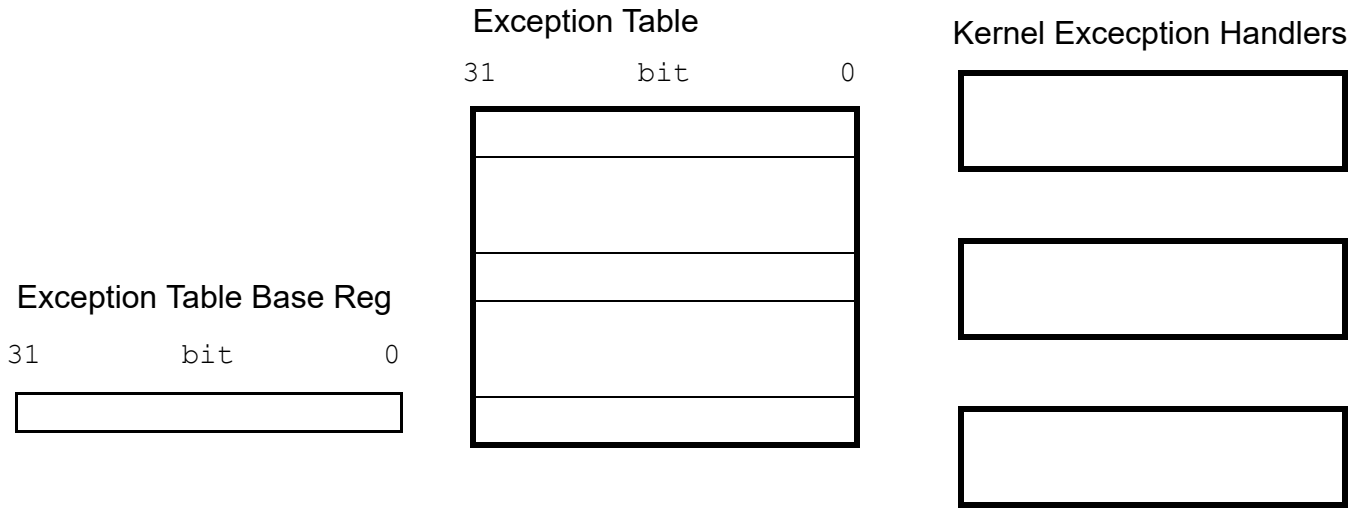
ETBR is for exception table base reg

ENUM is for exception number

EHA is for exception handler's address

Exception Table

exception number



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:

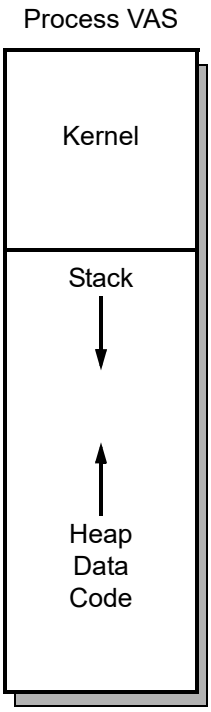
```
   .section .data
   string:
      .ascii "hello world\n"
   string_end:
      .equ len, string_end - string
   .section .text
   .global main
   main:
      movl $4, %eax
      movl $1, %ebx
      movl $string, %ecx
      movl $len, %edx
      int $0x80
      movl $1, %eax
      movl $0, %ebx
 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

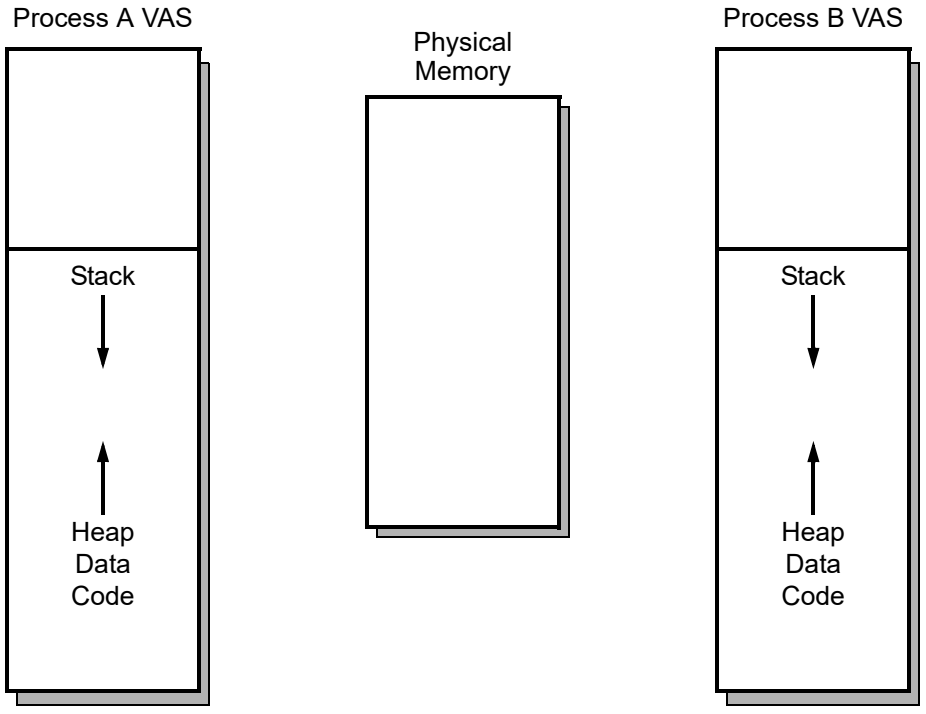
kernel mode

user mode

flipping modes

-
-
-

Sharing the Kernel



Context Switch

What? A *context switch*

-
-

When?

Why?

How?

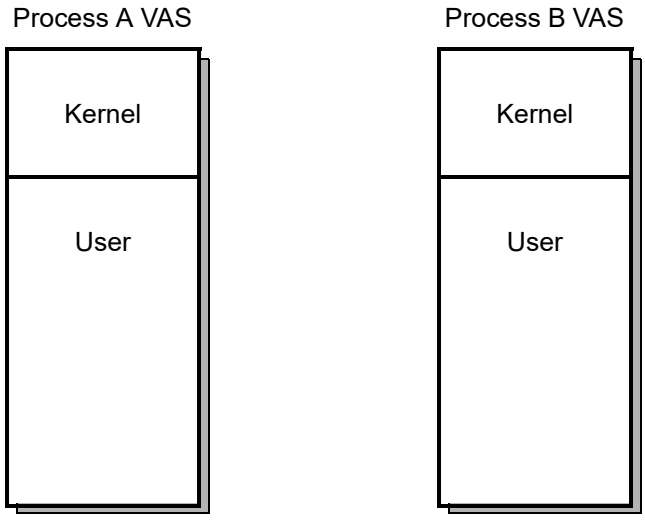
1.
2.
3.

❋ *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.