last time

make and Makefiles

- target: prereq (newline)(tab) commands
- suffix/pattern rules
- variables CC/CFLAGS/...

kernel mode versus user mode

- limit operations to OS code
- OS code checks "is this allowed"

system calls

- controlled entry into kernel mode
- starts at OS-specified location
- typically called by library (not directly)

on the lab

some common issues TAs/I saw:

- not checking that the `guesser` program worked
- setting CFLAGS, LDFLAGS, but not using them in rules
- wrong target first in Makefile (so ‘make’ doesn’t do ‘make all’)
- not setting either LD_LIBRARY_PATH (runtime) or -rpath (linktime)
- uploading files with spaces instead of tabs (copy/paste?)

misc. weirdness:

- apparently some versions of clang on portal may be missing libraries for `-static`?

quiz demo

warmup assignment

things programs on portal shouldn’t do

read other user’s files

modify OS’s memory

read other user’s data in memory

hang the entire system

things programs on portal shouldn’t do

read other user’s files

modify OS’s memory

read other user’s data in memory

hang the entire system

memory protection

reading from another program's memory?

Program A

```
0x10000: .word 42
     // ...
     // do work
     // ...
     movq 0x10000, %rax
```

Program B

```
// while A is working:
movq $99, %rax
movq %rax, 0x10000
...
```

memory protection

reading from another program's memory?

Program A	Program B

0x10000: .word 42
    // ...
    // do work
    // ...
    movq 0x10000, %rax
	// while A is working:
	movq $99, %rax
	movq %rax, 0x10000
	...

result: %rax (in A) is ...

A. 42 B. 99 C. 0x10000

D. 42 or 99 (depending on timing/program layout/etc)

E. 42 or 99 or program might crash (depending on ...)

F. something else

memory protection

reading from another program's memory?

Program A

```
0x10000: .word 42
      // ...
      // do work
      // ...
      movq 0x10000, %rax
```

Program B

```
// while A is working:
movq $99, %rax
movq %rax, 0x10000
...
```

result: `%rax` (in A) is 42 (always)

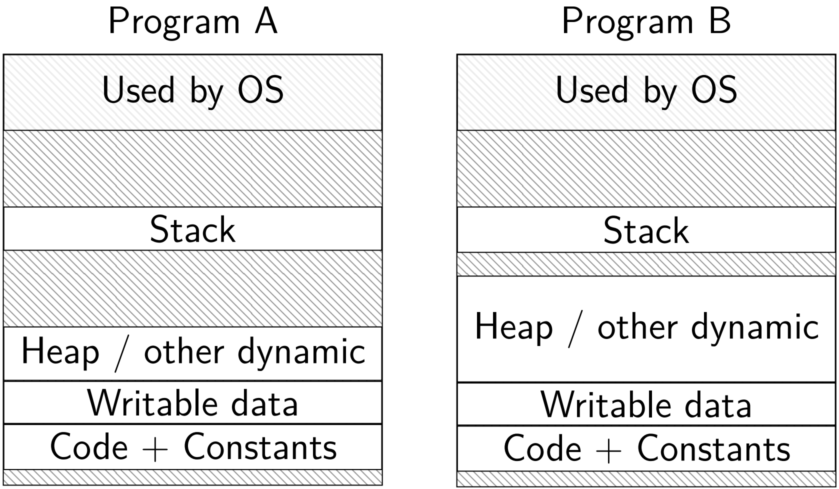
A. 42 B. 99 C. 0x10000

D. 42 or 99 (depending on timing/program layout/etc)

E. 42 or 99 or program might crash (depending on ...)

F. something else

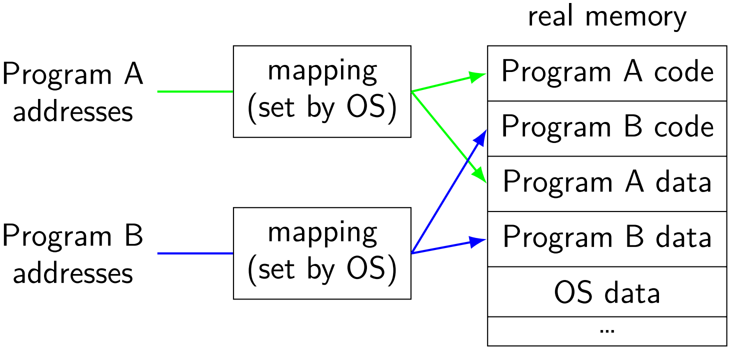
program memory (two programs)



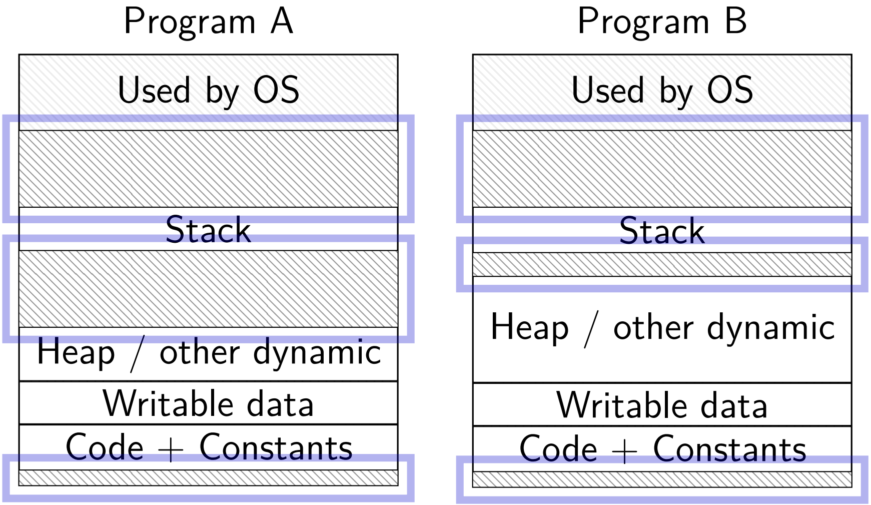
address space

programs have illusion of own memory

called a program's address space



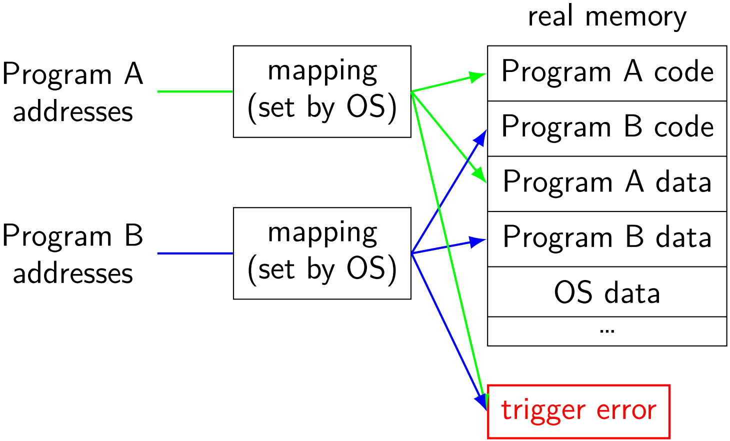
program memory (two programs)



address space

programs have illusion of own memory

called a program's address space



address space mechanisms

topic after exceptions

called virtual memory

mapping called page tables

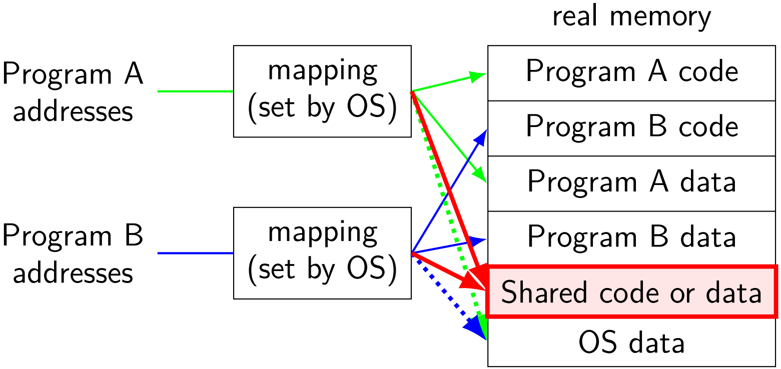
mapping part of what is changed in context switch

shared memory

recall: dynamically linked libraries

would be nice not to duplicate code/data...

we can!



one way to set shared memory on Linux

```
/* regular file, OR: */
int fd = open("/tmp/somefile.dat", O_RDWR);
/* special in-memory file */
int fd = shm_open("/name", O_RDWR);
...
/* make file's data accessible as memory */
void *memory = mmap(NULL, size, PROT_READ | PROT_WRITE,
                    MAP_SHARED, fd, 0);
```

mmap: "map" a file's data into your memory

will discuss a bit more when we talk about virtual memory

part of how Linux loads dynamically linked libraries

things programs on portal shouldn't do

read other user's files

modify OS's memory

read other user's data in memory

hang the entire system

an infinite loop

```
int main(void) {
    while (1) {
        /* waste CPU time */
    }
}
```

If I run this on a shared department machine, can you still use it?

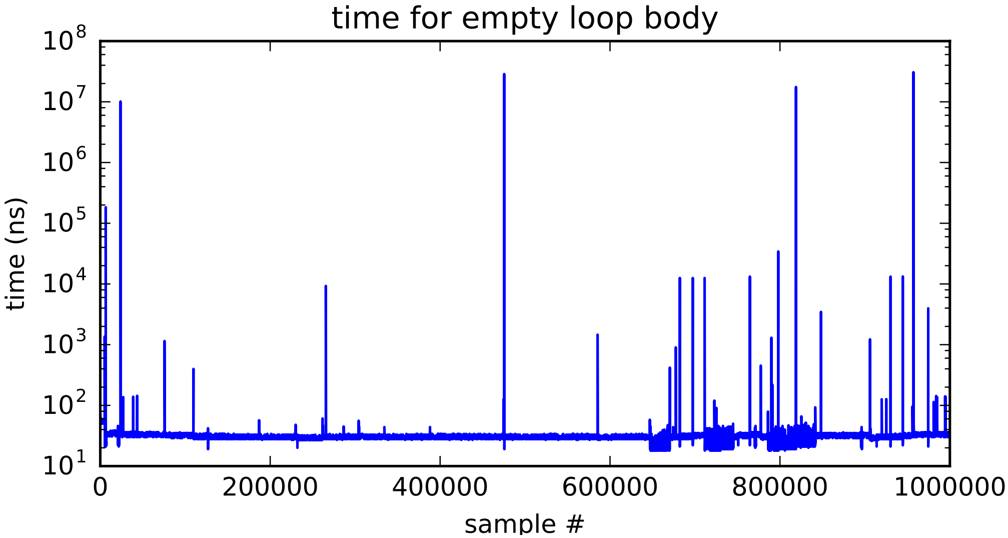
...if the machine only has one core?

timing nothing

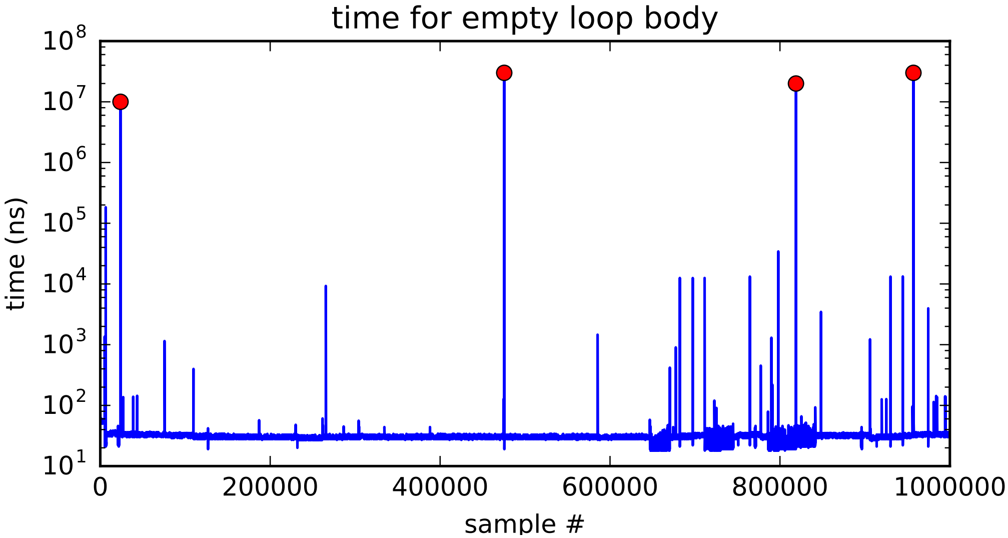
```
long times[NUM_TIMINGS];
int main(void) {
    for (int i = 0; i < N; ++i) {
        long start, end;
        start = get_time();
        /* do nothing */
        end = get_time();
        times[i] = end - start;
    }
    output_timings(times);
}
```

same instructions — same difference each time?

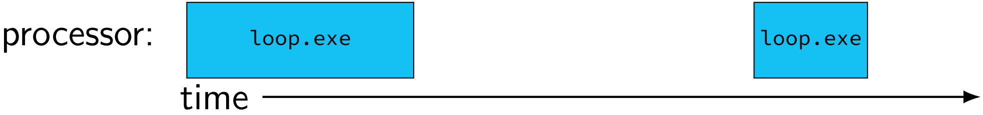
doing nothing on a busy system



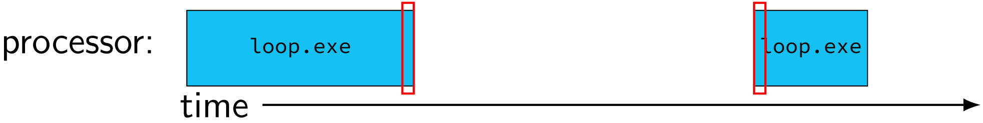
doing nothing on a busy system



time multiplexing



time multiplexing

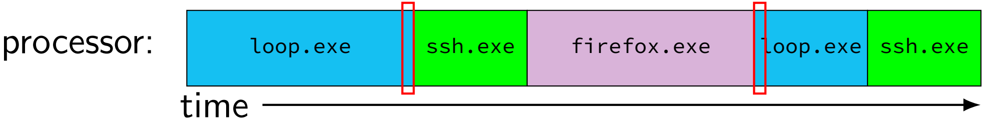


```
...
call get_time
    // whatever get_time does
movq %rax, %rbp
```

million cycle delay

```
call get_time
    // whatever get_time does
subq %rbp, %rax
...
```

time multiplexing

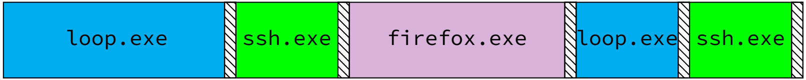


```
...
call get_time
    // whatever get_time does
movq %rax, %rbp
```

million cycle delay

```
call get_time
    // whatever get_time does
subq %rbp, %rax
...
```

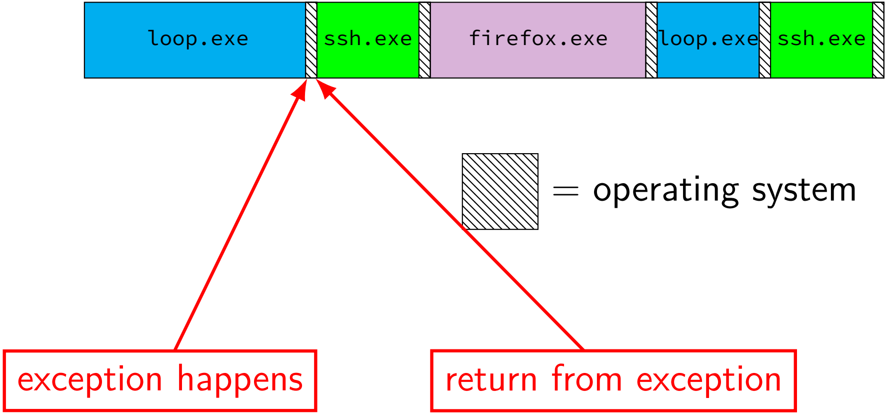
time multiplexing really



loop.exe ssh.exe firefox.exe loop.exe ssh.exe

▨ = operating system

time multiplexing really



▨ = operating system

exception happens

return from exception

threads

thread $=$ illusion of own processor

own register values

own program counter value

threads

thread = illusion of own processor

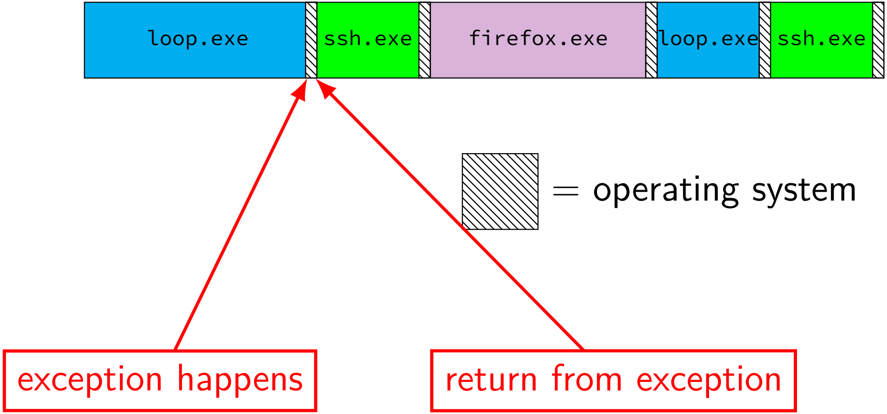
own register values

own program counter value

actual implementation:
many threads sharing one processor

- problem: where are register/program counter values when thread not active on processor?

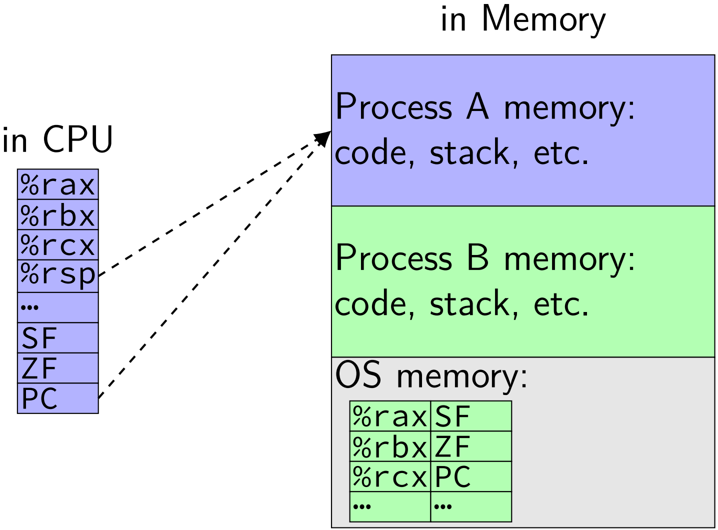
time multiplexing really



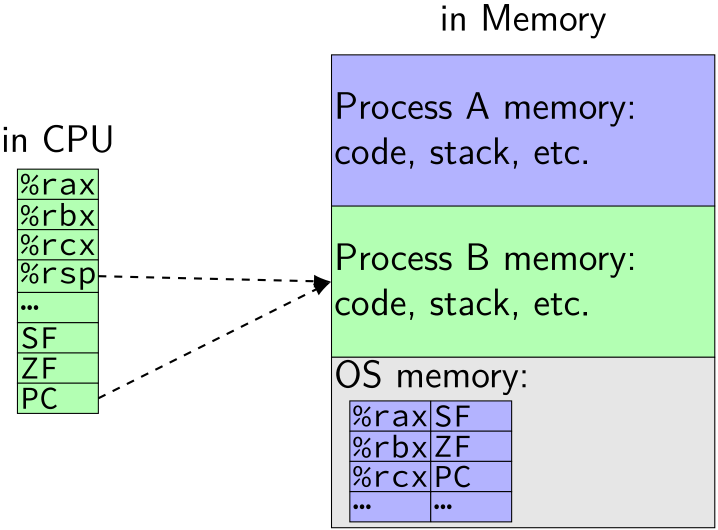
OS and time multiplexing

- starts running instead of normal program
  - mechanism for this: exceptions (later)
- saves old program counter, registers somewhere
- sets new registers, jumps to new program counter
- called context switch
  - saved information called context

contexts (A running)



contexts (B running)



in Memory

Process A memory:
code, stack, etc.

Process B memory:
code, stack, etc.

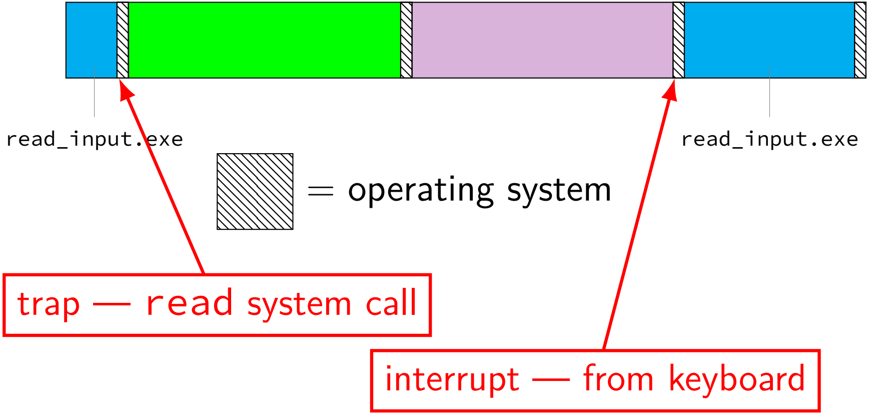
OS memory:

%rax	SF
%rbx	ZF
%rcx	PC
…	…

in CPU

- %rax
- %rbx
- %rcx
- %rsp
- …
- SF
- ZF
- PC

keyboard input timeline



types of exceptions

externally-triggered

- timer — keep program from hogging CPU
- I/O devices — key presses, hard drives, networks, …
- hardware is broken (e.g. memory parity error)

$\Big\}$ **asynchronous**
not triggered by running program

intentionally triggered exceptions

- system calls — ask OS to do something

errors/events in programs

- memory not in address space ("Segmentation fault")
- privileged instruction
- divide by zero
- invalid instruction

$\Big\}$ **synchronous**
triggered by current program

terms for exceptions

terms for exceptions aren't standardized

our readings use one set of terms

interrupts = externally-triggered
faults = error/event in program
trap = intentionally triggered

all these terms appear differently elsewhere

exception implementation

detect condition (program error or external event)

save current value of PC somewhere

jump to exception handler (part of OS)

- jump done without program instruction to do so

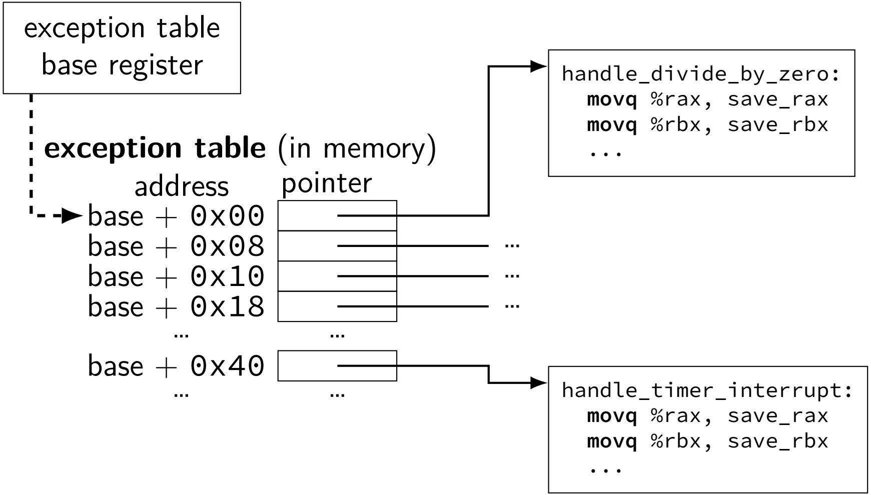
exception implementation: notes

I describe a simplified version

real x86/x86-64 is a bit more complicated

(mostly for historical reasons)

locating exception handlers



exception table base register

exception table (in memory)

address	pointer
base + `0x00`	
base + `0x08`	...
base + `0x10`	...
base + `0x18`	...
...	...
base + `0x40`	
...	...

```
handle_divide_by_zero:
  movq %rax, save_rax
  movq %rbx, save_rbx
  ...
```

```
handle_timer_interrupt:
  movq %rax, save_rax
  movq %rbx, save_rbx
  ...
```

running the exception handler

hardware saves the old program counter (and maybe more)

identifies location of exception handler via table

then jumps to that location

OS code can save anything else it wants to , etc.

which of these require exceptions? context switches?

A. program calls a function in the standard library

B. program writes a file to disk

C. program A goes to sleep, letting program B run

D. program exits

E. program returns from one function to another function

F. program pops a value from the stack

which require exceptions [answers] (1)

A. program calls a function in the standard library

no (same as other functions in program; some standard library functions might make system calls, but if so, that'll be part of what happens after they're called and before they return)

B. program writes a file to disk

yes (requires kernel mode only operations)

C. program A goes to sleep, letting program B run

yes (kernel mode usually required to change the address space to acess program B's memory)

which require exceptions [answer] (2)

D. program exits

yes (requires switching to another program, which requires accessing OS data + other program's memory)

E. program returns from one function to another function

no

F. program pops a value from the stack

no

which require context switches [answer]

no: A. program calls a function in the standard library

no: B. program writes a file to disk
(but might be done if program needs to wait for disk and other things could be run while it does)

yes: C. program A goes to sleep, letting program B run

yes: D. program exits

no: E. program returns from one function to another function

no: F. program pops a value from the stack

The Process

process = thread(s) + address space

illusion of dedicated machine:

- thread = illusion of own CPU
- address space = illusion of own memory

signals

Unix-like operating system feature

like exceptions for processes:

can be triggered by external process

- kill command/system call

can be triggered by special events

- pressing control-C
- other events that would normal terminate program
  - 'segmentation fault'
  - illegal instruction
  - divide by zero

can invoke signal handler (like exception handler)

exceptions v signals

(hardware) exceptions	signals
handler runs in kernel mode	handler runs in user mode
hardware decides when	OS decides when
hardware needs to save PC	OS needs to save PC + registers
processor next instruction changes	thread next instruction changes

exceptions v signals

(hardware) exceptions	signals
handler runs in kernel mode	handler runs in user mode
hardware decides when	OS decides when
hardware needs to save PC	OS needs to save PC + registers
processor next instruction changes	thread next instruction changes

…but OS needs to run to trigger handler
most likely "forwarding" hardware exception

exceptions v signals

(hardware) exceptions	signals
handler runs in kernel mode	handler runs in user mode
hardware decides when	OS decides when
hardware needs to save PC	OS needs to save PC + registers
processor next instruction changes	thread next instruction changes

signal handler follows normal calling convention
not special assembly like typical exception handler

exceptions v signals

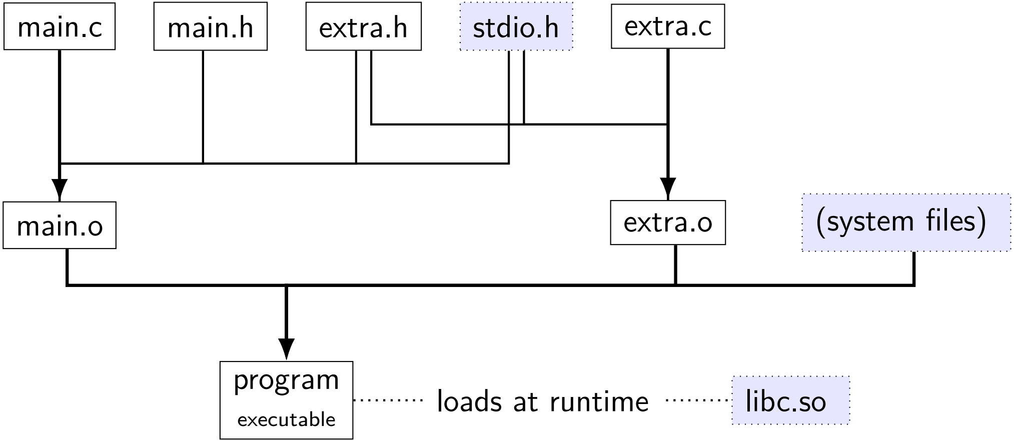
(hardware) exceptions	signals
handler runs in kernel mode	handler runs in user mode
hardware decides when	OS decides when
hardware needs to save PC	OS needs to save PC + registers
processor next instruction changes	thread next instruction changes

signal handler runs in same thread ('virtual processor') as process was using before

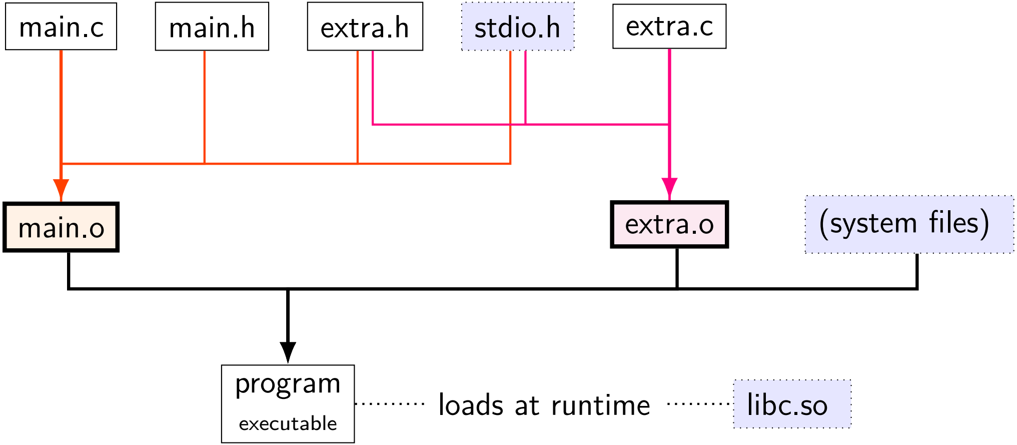
not running at 'same time' as the code it interrupts

backup slides

files in building C programs [dynamic linking]

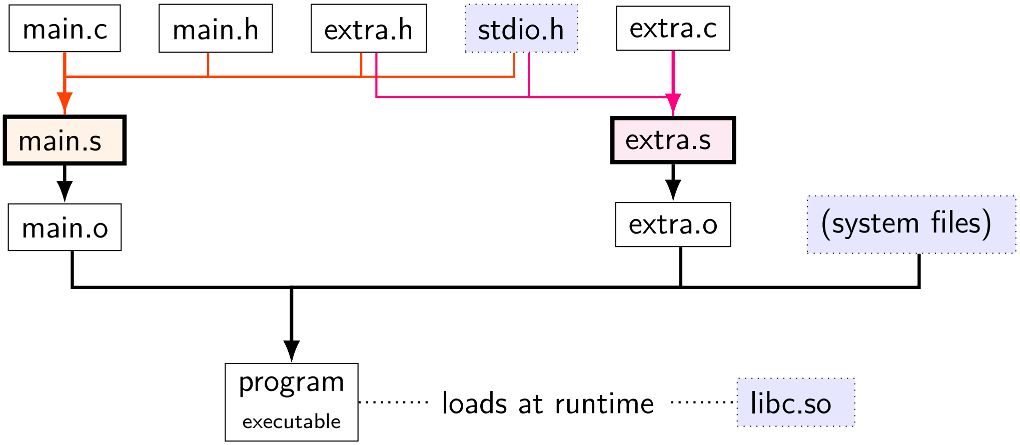


files in building C programs [dynamic linking]



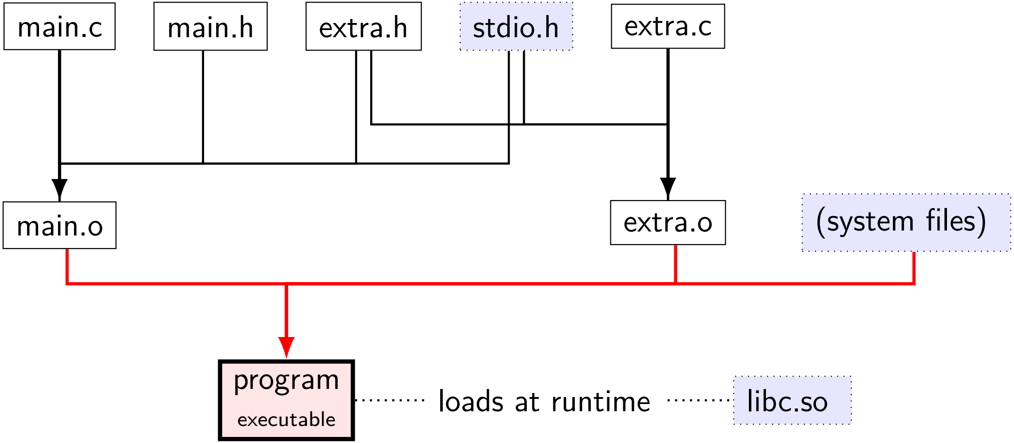
```
clang -c main.c
clang -c extra.c
```

files in building C programs [dynamic linking]



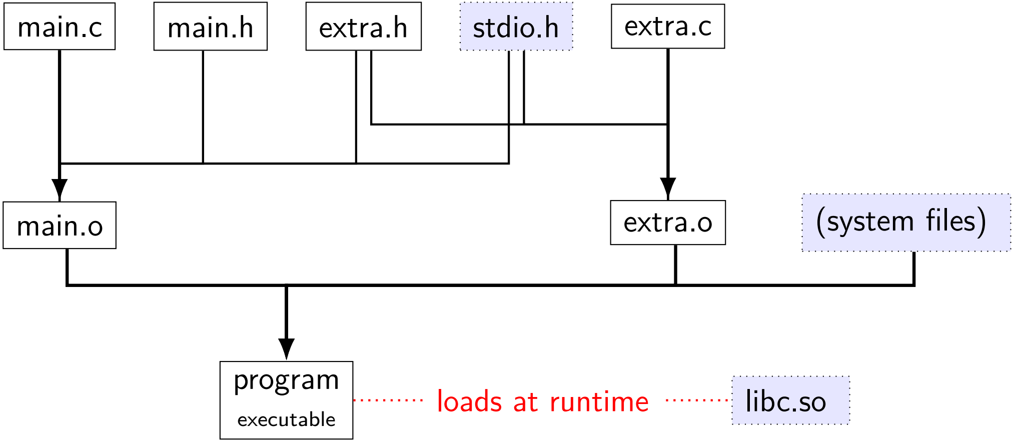
```
clang -S -c main.c
clang -S -c extra.c
```

files in building C programs [dynamic linking]



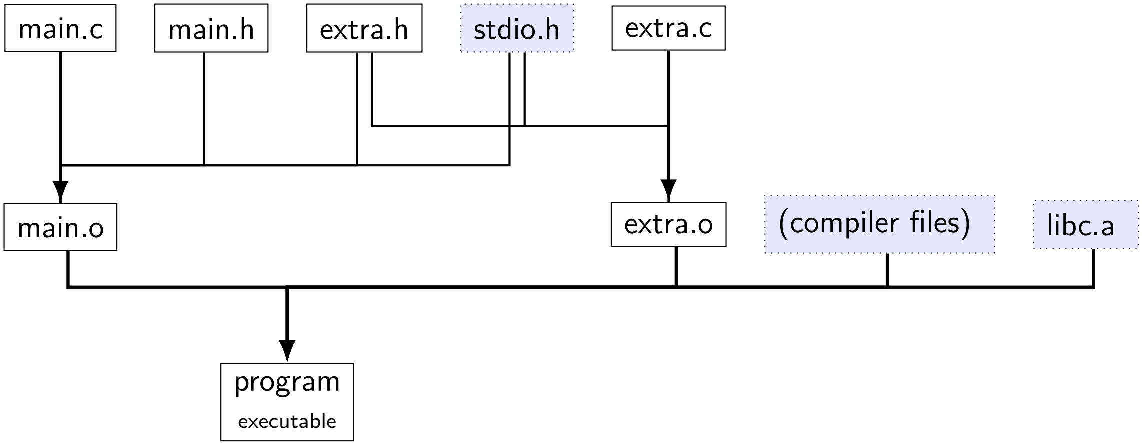
```
clang -o program main.o extra.o
```

files in building C programs [dynamic linking]



```
./program ...
```

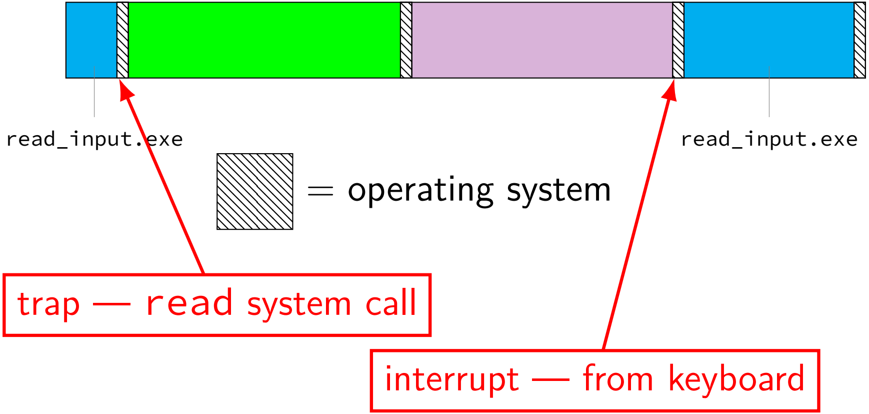
files in building C programs [static linking]



file extensions

name		
`.c`		C source code
`.h`		C header file
`.s`	(or `.asm`)	assembly file
`.o`	(or `.obj`)	object file (binary of assembly)
(none)	(or `.exe`)	executable file
.a	(or `.lib`)	statically linked library [collection of .o files]
.so	(or `.dll`)	dynamically linked library ['shared object']

keyboard input timeline

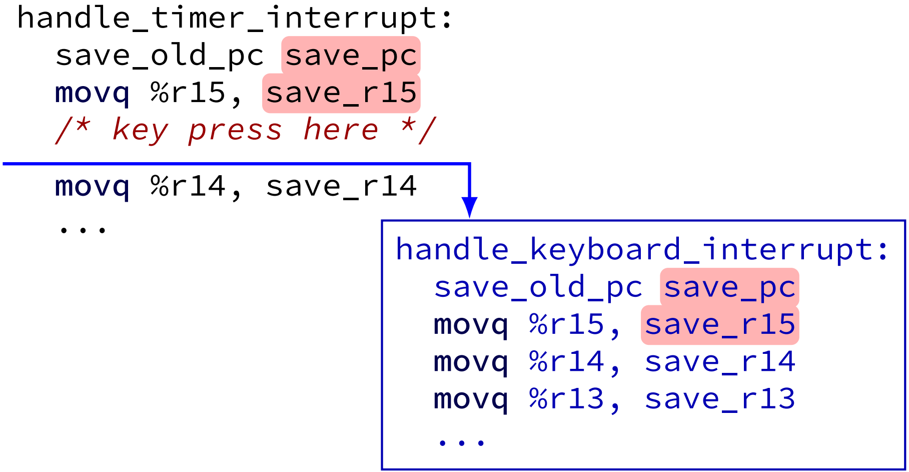


exceptions in exceptions

```
handle_timer_interrupt:
  save_old_pc save_pc
  movq %r15, save_r15
  /* key press here */

  movq %r14, save_r14
  ...
```

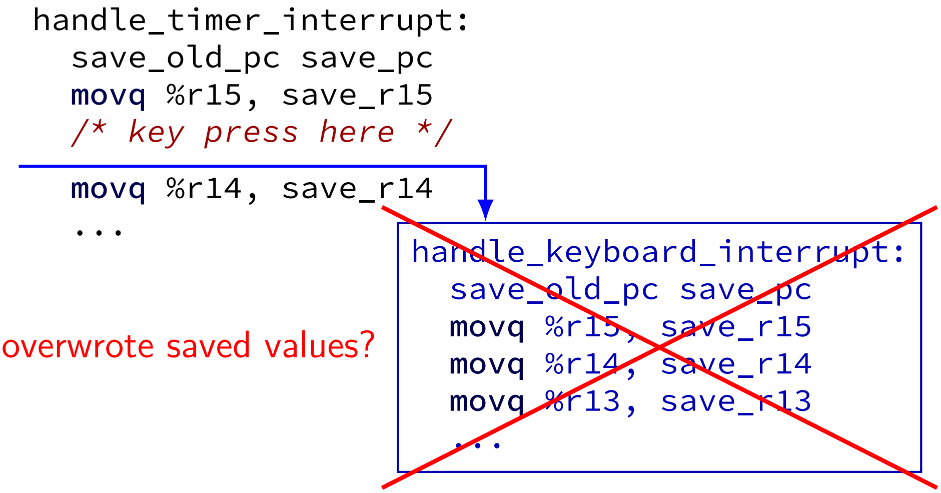
exceptions in exceptions



```
handle_timer_interrupt:
  save_old_pc save_pc
  movq %r15, save_r15
  /* key press here */
  movq %r14, save_r14
  ...
```

```
handle_keyboard_interrupt:
  save_old_pc save_pc
  movq %r15, save_r15
  movq %r14, save_r14
  movq %r13, save_r13
  ...
```

exceptions in exceptions



```
handle_timer_interrupt:
  save_old_pc save_pc
  movq %r15, save_r15
  /* key press here */
  movq %r14, save_r14
  ...
```

```
handle_keyboard_interrupt:
  save_old_pc save_pc
  movq %r15, save_r15
  movq %r14, save_r14
  movq %r13, save_r13
  ...
```

oops, overwrote saved values?

interrupt disabling

CPU supports disabling (most) interrupts

interrupts will wait until it is reenabled

CPU has extra state:

- are interrupts enabled?
- is keyboard interrupt pending?
- is timer interrupt pending?

exceptions in exceptions

```
handle_timer_interrupt:
  /* interrupts automatically disabled here */
  movq %rsp, save_rsp
  save_old_pc save_pc
  /* key press here */
  jmpIfFromKernelMode skip_exception_stack
  movq current_exception_stack, %rsp
skip_set_kernel_stack:
  pushq save_rsp
  pushq save_pc
  enable_intterupts2
  pushq %r15
  ...
  /* interrupt happens here! */
  ...
```

exceptions in exceptions

```
handle_timer_interrupt:
  /* interrupts automatically disabled here */
  movq %rsp, save_rsp
  save_old_pc save_pc
  /* key press here */
  jmpIfFromKernelMode skip_exception_stack
  movq current_exception_stack, %rsp
skip_set_kernel_stack:
  pushq save_rsp
  pushq save_pc
  enable_intterupts2
  pushq %r15
  ...
  /* interrupt happens here! */
  ...
```

exceptions in exceptions

```
handle_timer_interrupt:
  /* interrupts automatically disabled here */
  movq %rsp, save_rsp
  save_old_pc save_pc
  /* key press here */
  jmpIfFromKernelMode skip_exception_stack
  movq current_exception_stack, %rsp
skip_set_kernel_stack:
  pushq save_rsp
  pushq save_pc
  enable_intterupts2
  pushq %r15
  ...
  /* interrupt happens here! */
  ...
```

```
handle_keyboard_interrupt:
  movq %rsp, save_rsp
```

disabling interrupts

automatically disabled when exception handler starts

also can be done with privileged instruction:

```
change_keyboard_parameters:
  disable_interrupts
  ...
  /* change things used by
     handle_keyboard_interrupt here */
  ...
  enable_interrupts
```