CS3281 / CS5281

Exceptional Control Flow

Will Hedgecock
Sandeep Neema
Bryan Ward

**Some lecture slides borrowed and adapted from CMU's "Computer Systems: A Programmer's Perspective"*

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

VANDERBILT UNIVERSITY

This Lecture

- **Exceptional Control Flow**
- Exceptions
- Processes
- Process Control

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Control Flow

- Processors do only one thing:
  - From startup to shutdown, a CPU reads and executes (interprets) a sequence of instructions, one at a time
  - This sequence is the CPU's *control flow* (or *flow of control*)

Time

Physical control flow

<startup>

$inst_1$

$inst_2$

$inst_3$

…

$inst_n$

<shutdown>

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

VANDERBILT UNIVERSITY

Altering the Control Flow

- You know two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return
- Insufficient for a useful system: difficult to <u>react</u> to events and changing system state
  - Examples of changes in system state:
    - Data arrives from a disk or a network adapter
    - Instruction divides by zero
    - User hits Ctrl-C at the keyboard
    - System timer expires
- System needs mechanisms for “exceptional control flow”

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu
VANDERBILT UNIVERSITY

Exceptional Control Flow

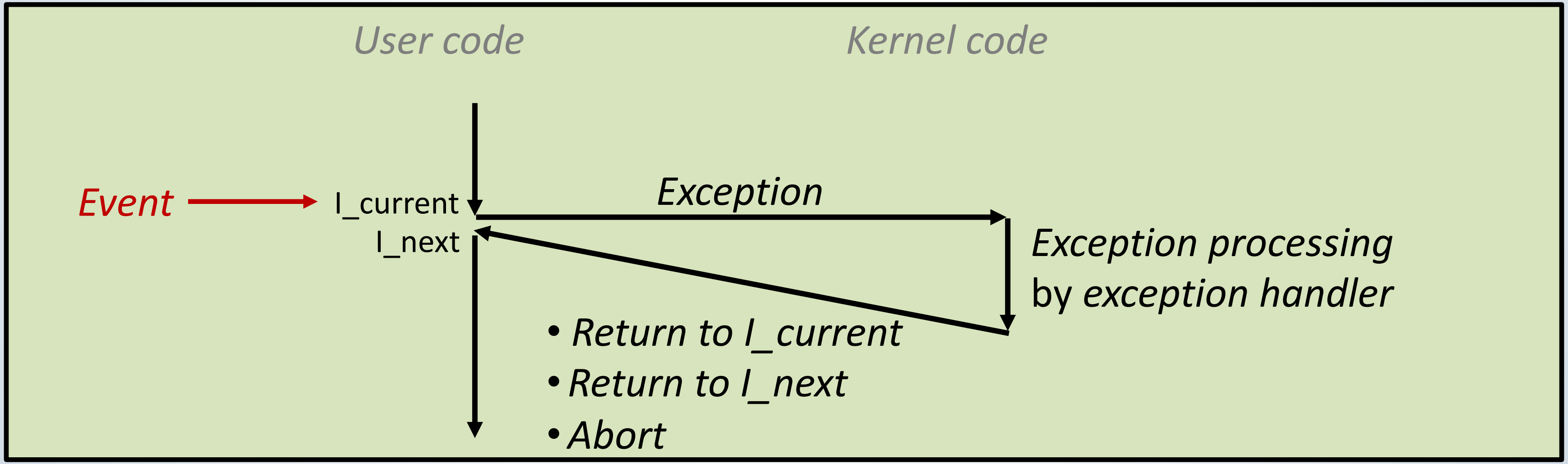
- Exists at all levels of a computer system
- Low-level mechanisms:
  - **Exceptions**: change in control flow in response to a system event
    - Implemented using combination of hardware and OS software
- High-level mechanisms:
  - **Process context switch**, i.e., stop running one program and start running another
    - Implemented by OS software and hardware timer
  - **Signals:** a means of sending a (limited) message to a process that they should respond to
    - Implemented by OS software
  - **Nonlocal jumps**: setjmp() and longjmp()
    - Implemented by C runtime library

This Lecture

- Exceptional Control Flow
- **Exceptions**
- Processes
- Process Control

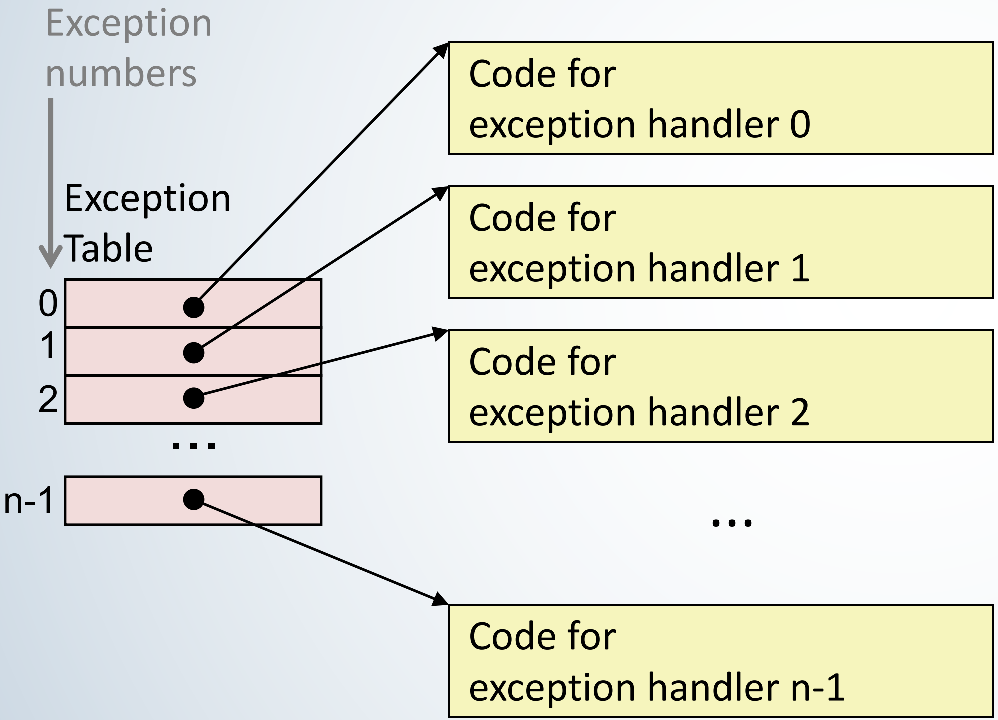
Exceptions

- An exception is a transfer of control to the OS kernel in response to some event (i.e., change in processor state)
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Exception Tables



- Each type of event has a unique exception number k
- k = index into exception table (a.k.a. interrupt vector)
- Handler k is called each time exception k occurs

Asynchronous Exceptions

- Caused by events external to the processor
  - Indicated by setting the processor's interrupt pin
  - Handler returns to "next" instruction
- Examples:
  - Timer interrupt
    - Every few ms, a timer chip trigger an interrupt
    - Used by kernel to take back control from user programs
  - I/O interrupt from external device
    - Arrival of network packet
    - Arrival of data from disk

Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - Traps
    - Intentional
    - Examples: syscalls, breakpoint traps, special instructions
    - Returns control to “next” instruction
  - Faults
    - Unintentional but possibly recoverable
    - Examples: page faults (recoverable), protection faults (unrecoverable), floating-point exceptions
    - Either re-execute faulting instruction or abort
  - Aborts
    - Unintentional and unrecoverable
    - Examples: illegal instruction, parity error, machine check
    - Aborts current program

System Calls

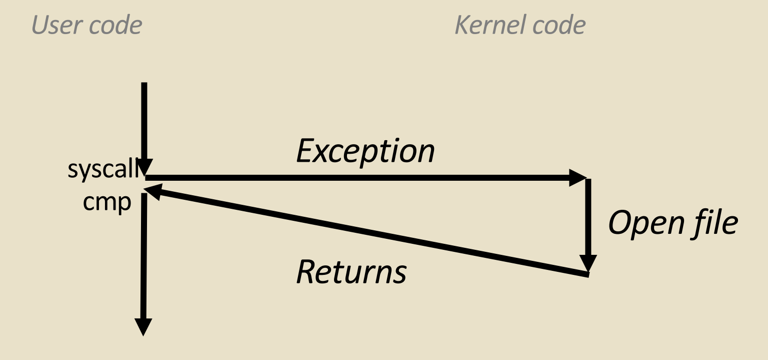
- Each system call has a unique ID number
- xv6 syscall numbers defined in kernel/syscall.h
- Linux has many more system calls as it is much more complex
- You will be implementing your own system calls, so you will have to add to this

```c
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
```

System Call Example: Opening File

- User calls: `open(filename, options)`
- Calls `__open` function, which invokes system call instruction `syscall`

```
00000000000e5d70 <__open>:
...
e5d79:   b8 02 00 00 00      mov  $0x2,%eax  # open is syscall #2
e5d7e:   0f 05               syscall         # Return value in %rax
e5d80:   48 3d 01 f0 ff ff   cmp  $0xfffffffffffff001,%rax
...
e5dfa:   c3                  retq
```



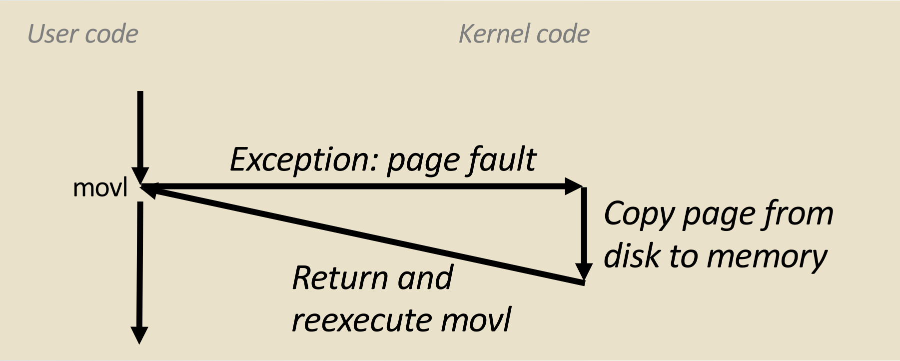
- `%eax` contains syscall number
- Other arguments in `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9`
- Return value in `%rax`
- Negative value is an error corresponding to negative `errno`

Fault Example: Page Fault

- User writes to memory location
- That portion (page) of user's memory is currently on disk

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

```
80483b7: c7 05 10 9d 04 08 0d      movl    $0xd,0x8049d10
```

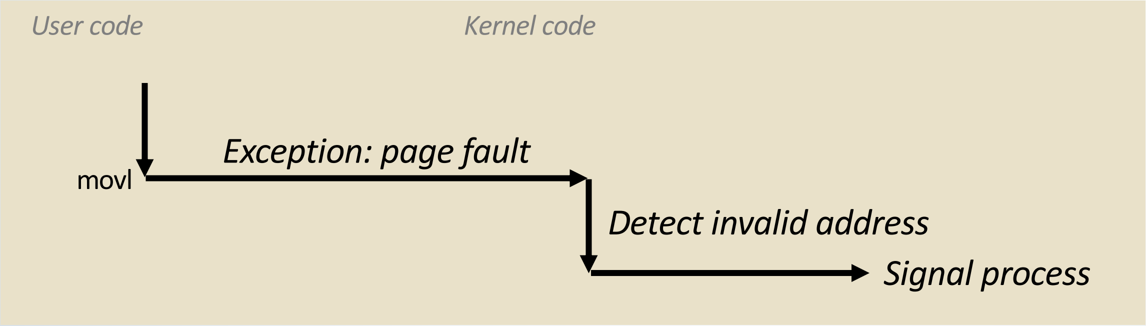


Fault Example: Invalid Memory Reference

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

- Sends `SIGSEGV` signal to user process
- User process exits with “segmentation fault”

```
80483b7: c7 05 60 e3 04 08 0d     movl    $0xd,0x804e360
```



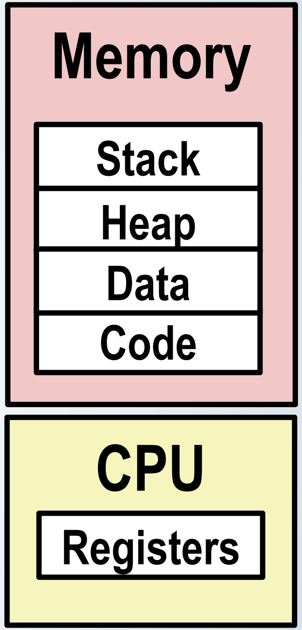
This Lecture

- Exceptional Control Flow
- Exceptions
- **Processes**
- Process Control

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

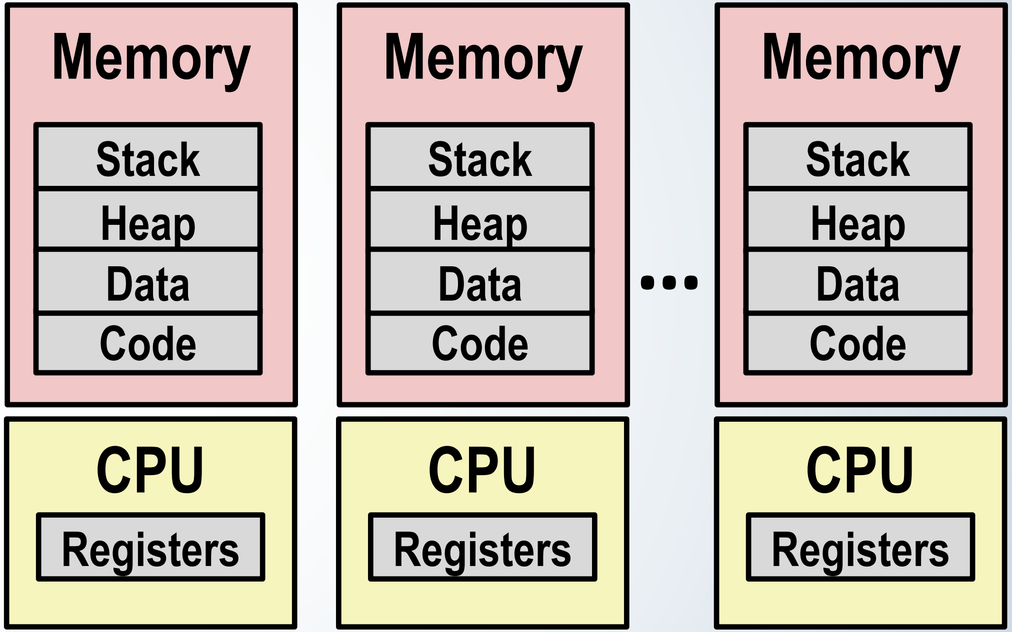
Processes

- Definition: A process is an instance of a running program
  - One of the most profound ideas in computer science
  - Not the same as “program” or “processor”
- Process provides each program with two key abstractions:
  - Logical control flow
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called context switching
  - Private address space
    - Each program seems to have exclusive use of main memory
    - Provided by kernel mechanism called virtual memory



Multiprocessing: The Illusion

- Computer runs many processes simultaneously
  - Applications for one or more users
    - Web browsers, email clients, editors, …
  - Background tasks
    - Monitoring network & I/O devices



Multiprocessing Example

- Running program “top” on Mac
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)

xterm

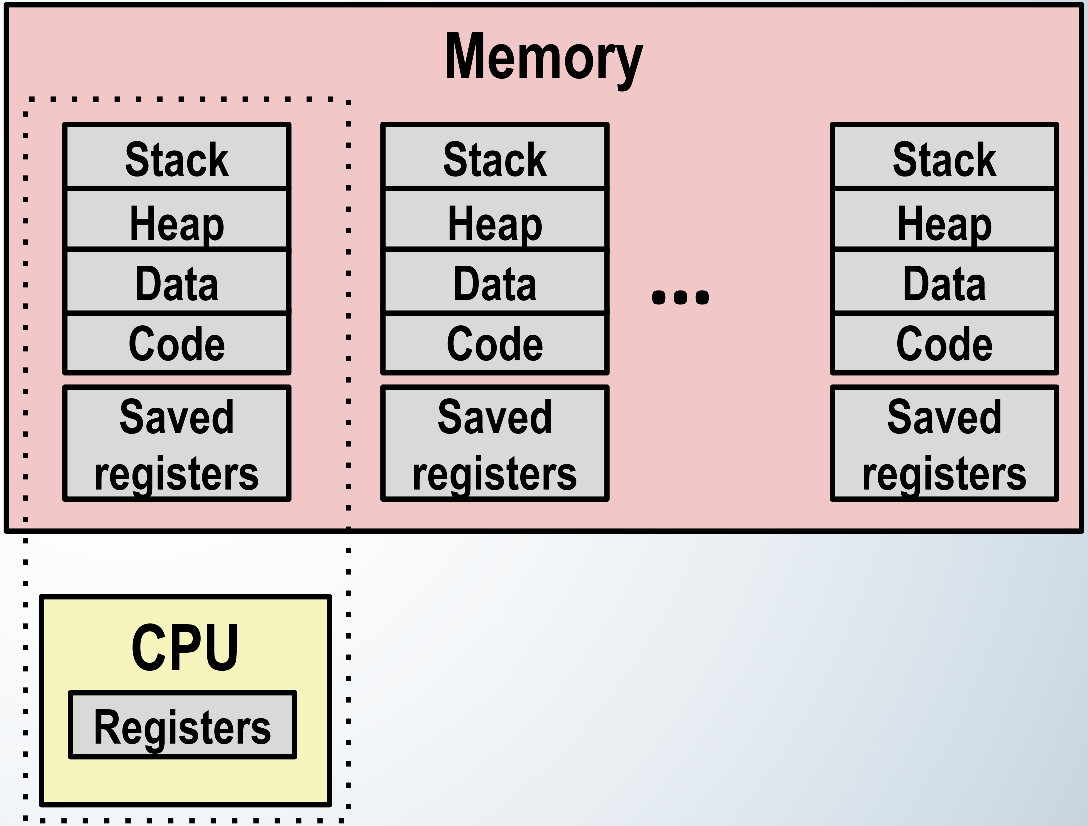
```
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads          11:47:07
Load Avg: 1.03, 1.13, 1.14  CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 576K resident, 0B data, 0B linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1039M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280G vsize, 1091M framework vsize, 23075213(1) pageins, 5843367(0) pageouts.
Networks: packets: 41046228/11G in, 66083096/77G out.
Disks: 17874391/349G read, 12847373/594G written.

PID    COMMAND      %CPU TIME     #TH  #WQ  #PORT #MREG RPRVT  RSHRD  RSIZE  VPRVT  VSIZE
99217- Microsoft Of 0.0  02:28.34 4    1    202   418   21M    24M    21M    66M    763M
99051  usbmuxd      0.0  00:04.10 3    1    47    66    436K   216K   480K   60M    2422M
99006  iTunesHelper 0.0  00:01.23 2    1    55    78    728K   3124K  1124K  43M    2429M
84286  bash         0.0  00:00.11 1    0    20    24    224K   732K   484K   17M    2378M
84285  xterm        0.0  00:00.83 1    0    32    73    656K   872K   692K   9728K  2382M
55939- Microsoft Ex 0.3  21:58.97 10   3    360   954   16M    65M    46M    114M   1057M
54751  sleep        0.0  00:00.00 1    0    17    20    92K    212K   360K   9632K  2370M
54739  launchdadd   0.0  00:00.00 2    1    33    50    488K   220K   1736K  48M    2409M
54737  top          6.5  00:02.53 1/1  0    30    29    1416K  216K   2124K  17M    2378M
54719  automountd   0.0  00:00.02 7    1    53    64    860K   216K   2184K  53M    2413M
54701  ocspd        0.0  00:00.05 4    1    61    54    1268K  2644K  3132K  50M    2426M
54661  Grab         0.6  00:02.75 6    3    222+  389+  15M+   26M+   40M+   75M+   2556M+
54659  cookied      0.0  00:00.15 2    1    40    61    3316K  224K   4088K  42M    2411M
53818  mdworker     0.0  00:01.67 4    1    52    91    7628K  7412K  16M    48M    2438M
50878  mdworker     0.0  00:11.17 3    1    53    91    2464K  6148K  9976K  44M    2434M
50410  xterm        0.0  00:00.13 1    0    32    73    280K   872K   532K   9700K  2382M
50078  emacs        0.0  00:06.70 1    0    20    35    52K    216K   88K    18M    2392M
```

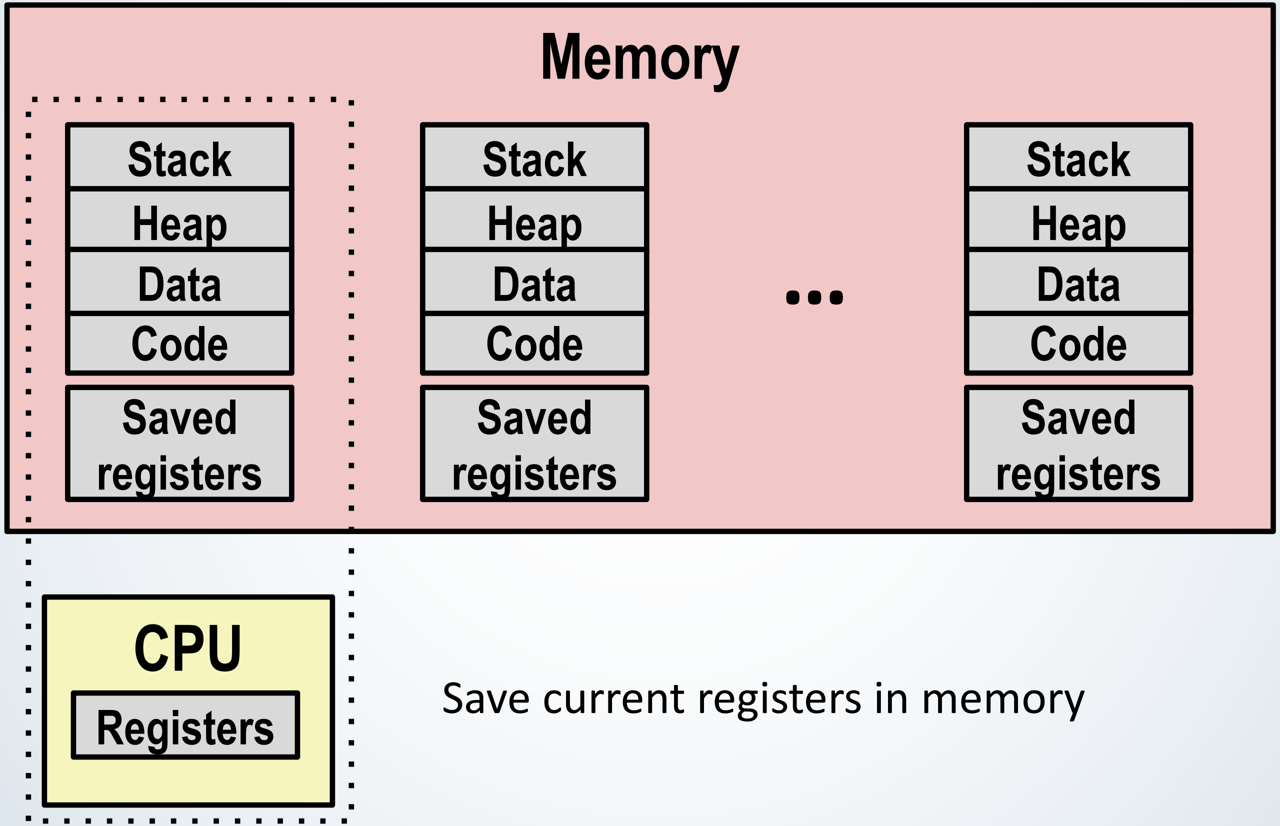
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Multiprocessing: The (Traditional) Reality

- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (later in course)
  - Register values for nonexecuting processes saved in memory

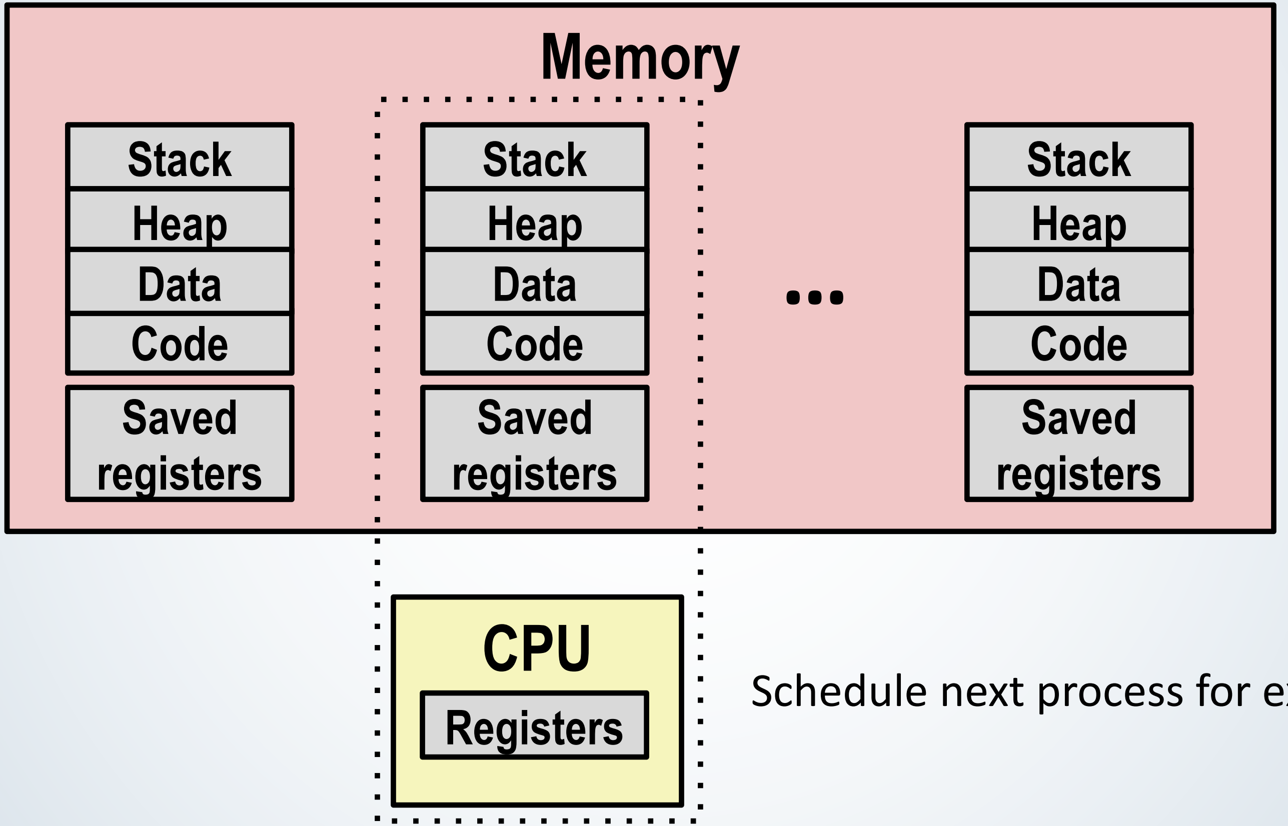


Multiprocessing: The (Traditional) Reality



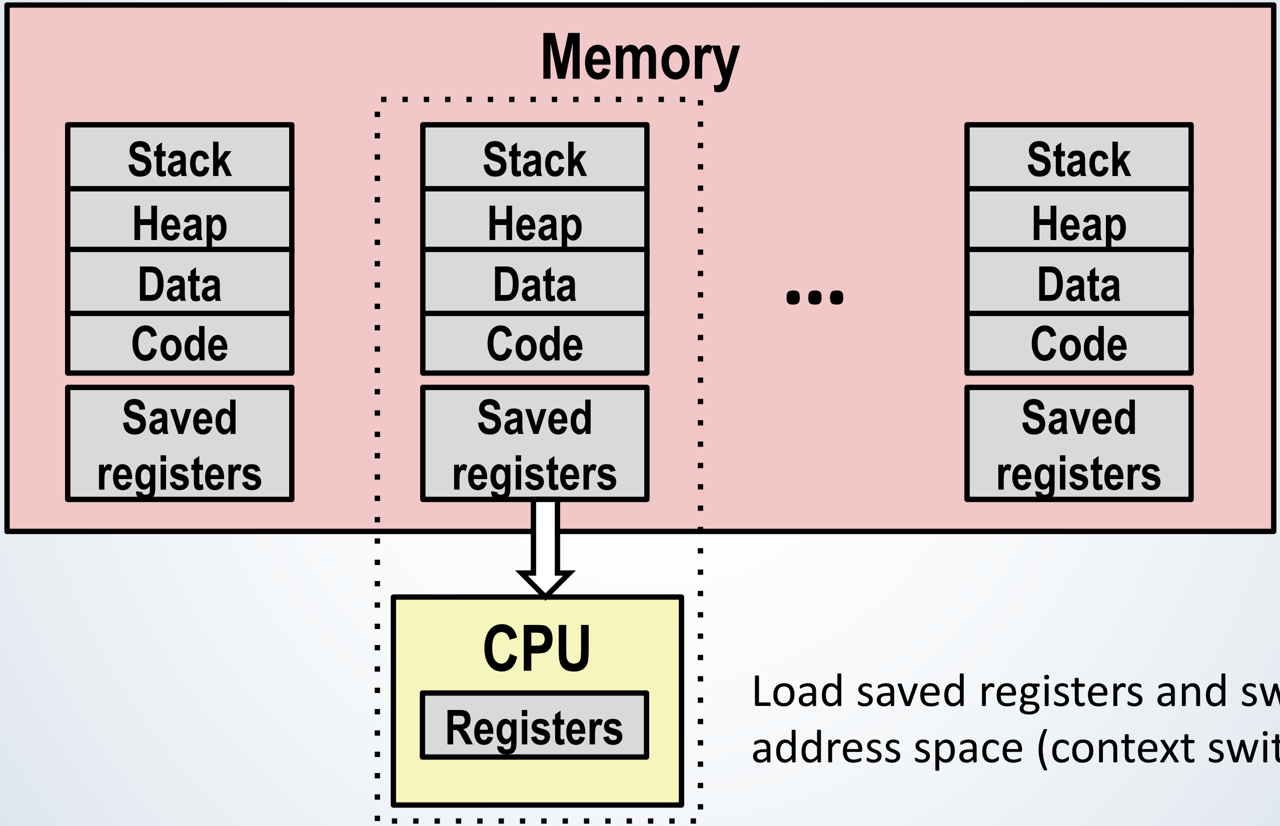
Save current registers in memory

Multiprocessing: The (Traditional) Reality



Schedule next process for execution

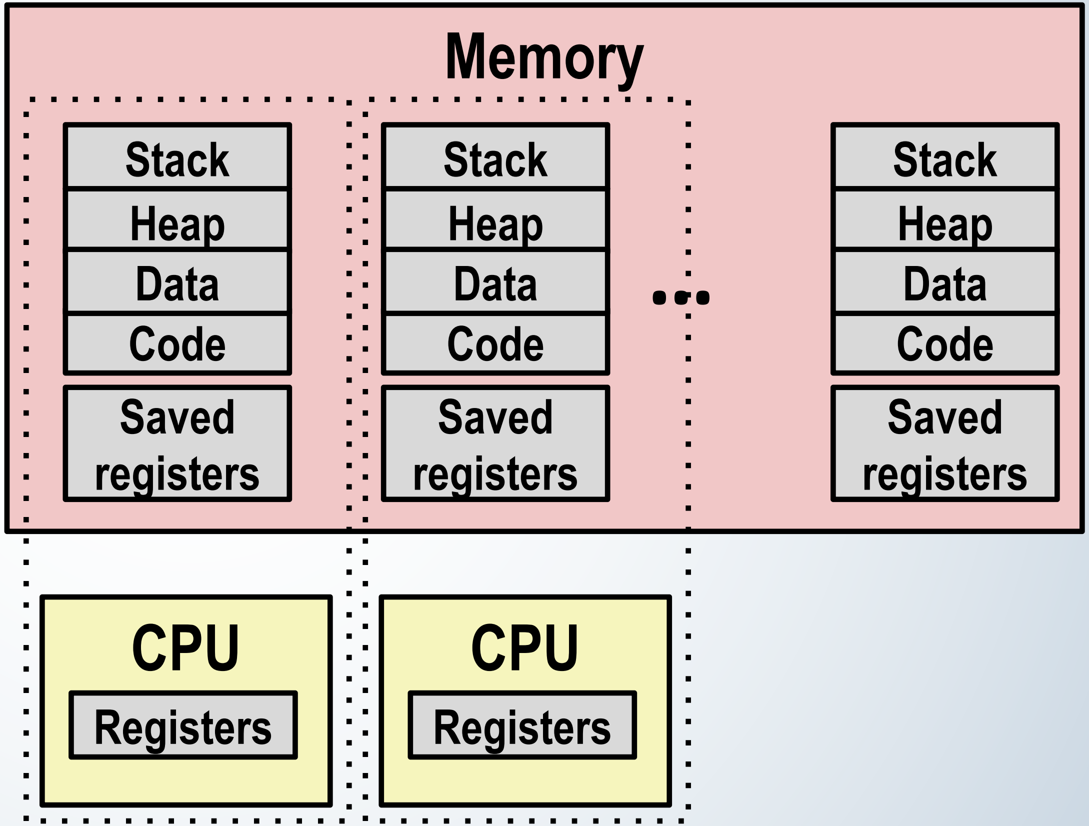
Multiprocessing: The (Traditional) Reality



Load saved registers and switch address space (context switch)

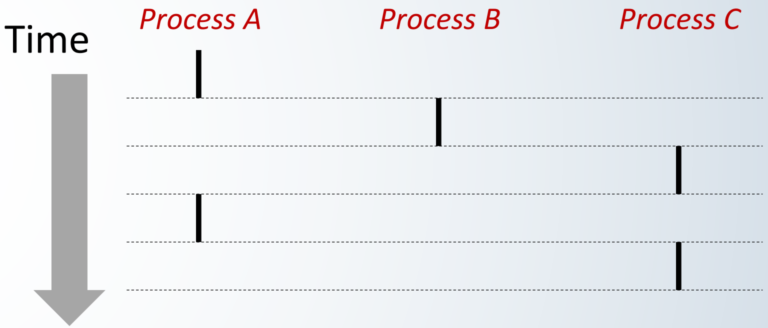
Multiprocessing: The (Modern) Reality

- Multicore processors
  - Multiple CPUs on single chip
  - Share main memory (and some of the caches)
  - Each can execute a separate process
  - Scheduling of processors onto cores done by kernel



Concurrent Processes

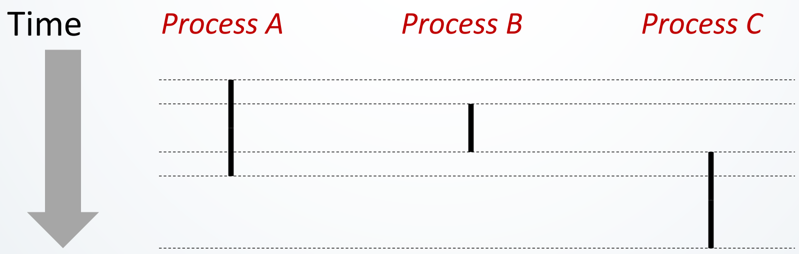
- Each process is a logical control flow
- Two processes *run* *concurrently* (*are concurrent)* if their flows overlap in time
- Otherwise, they are *sequential*
- Examples (running on single core):
  - Concurrent: A & B, A & C
  - Sequential: B & C



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

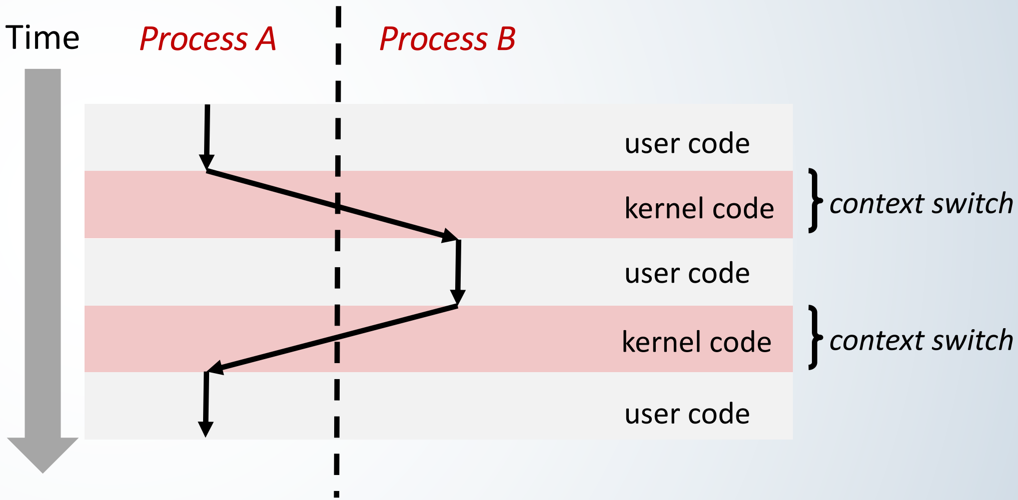
User View of Concurrent Processes

- Control flows for concurrent processes are physically disjoint in time
- However, we can think of concurrent processes as running in parallel with each other



Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the *kernel*
  - **Important:** the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a *context switch*



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu