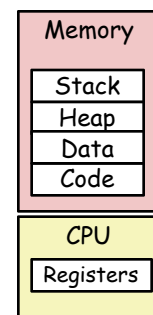


# Processes & Threads

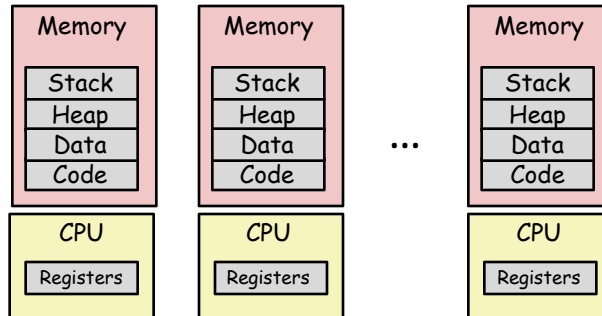
CS 475

## 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

```

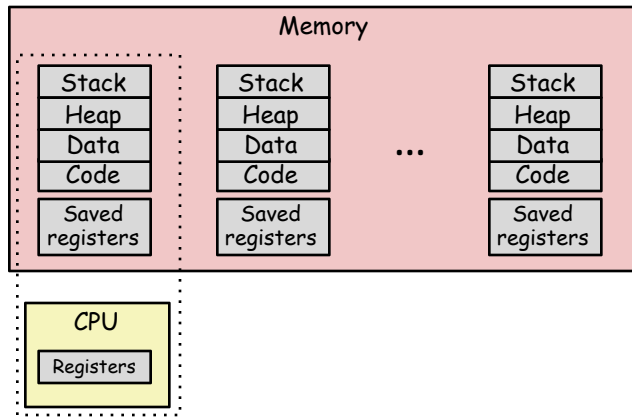
xterm
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads
Load Avg: 1.03, 1.13, 1.14 CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 575K resident, 0B data, 0B linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1033M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280K vsize, 1091M framework vsize, 23075213(1) pageins, 5843357(0) pageouts.
Network: 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 0.0 02:28.34 4 1 202 418 21M 24M 21M 66M 763M
99051- usbmuxd 0.0 00:04.10 3 1 47 56 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 21:58.37 10 3 350 954 16M 65M 48M 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 64 1268K 2644K 3132K 50M 2426M
54651- Grab 0.6 00:02.75 6 3 222+ 389+ 15M+ 26M+ 40M+ 75M+ 2556M+
54659- cookiec 0.0 00:00.15 2 1 40 61 3316K 224K 4088K 42M 2411M
53919- mdworker 0.0 00:01.57 4 1 52 91 7628K 7412K 16M 48M 2439M
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 215K 88K 18M 2392M

```

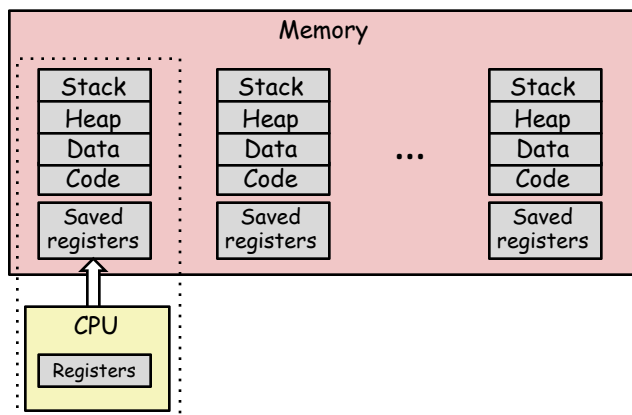
- ❑ Running program "top" on Mac or a Linux system
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)

## Multiprocessing: The (Traditional) Reality



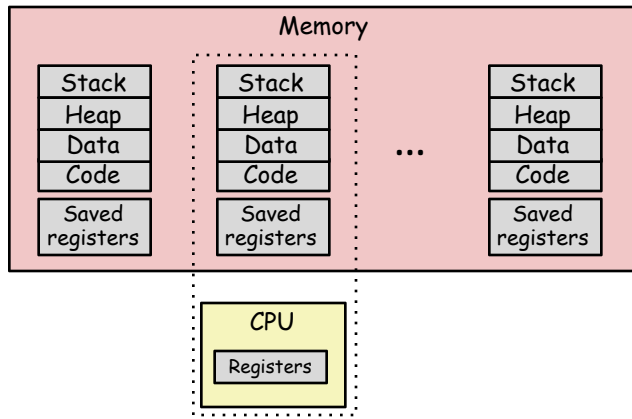
- ❑ Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (CS 471)
  - Register values for non-executing processes saved in memory

## Multiprocessing: The (Traditional) Reality



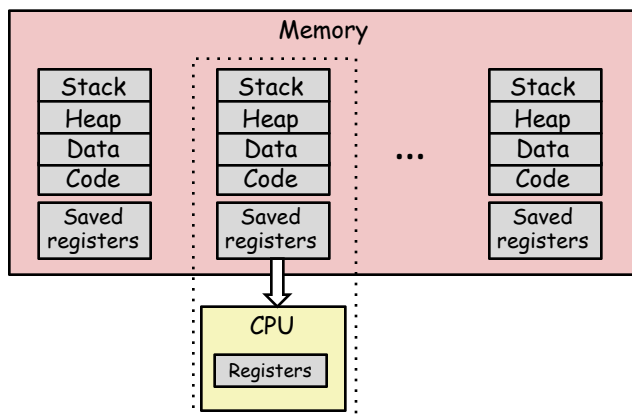
- ❑ Save current registers in memory

## Multiprocessing: The (Traditional) Reality



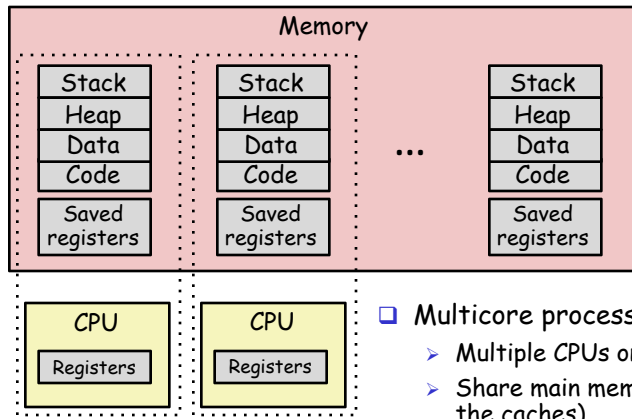
- ❑ Schedule next process for execution

## Multiprocessing: The (Traditional) Reality



- ❑ Load saved registers and switch address space (context switch)

## Multiprocessing: The (Modern) Reality

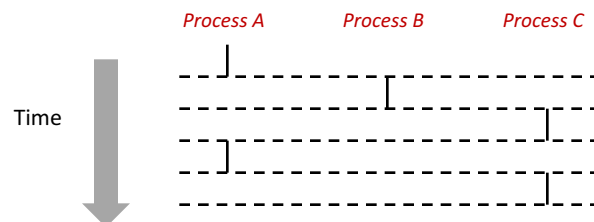


### ❑ Multicore processors

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

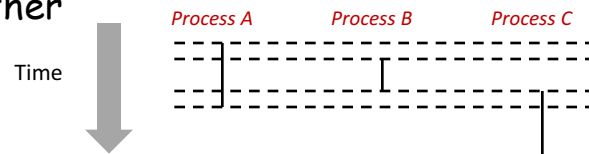
## Concurrent Processes

- ❑ 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



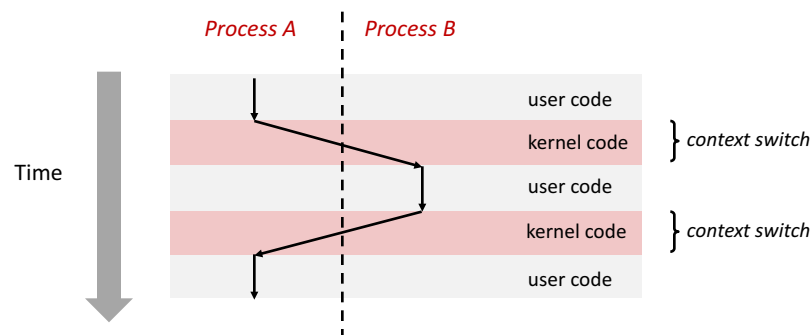
## User View of Concurrent Processes

- ❑ 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*



## Cooperating Concurrent Processes

- ❑ Concurrent processes part of the same application
- ❑ Processes "cooperate" on task
- ❑ Motivation
  - Support inherent concurrency in application
    - Window systems, web servers
  - Improved performance - can make use of multiple processors

CS 475

13

## Concurrent Programs

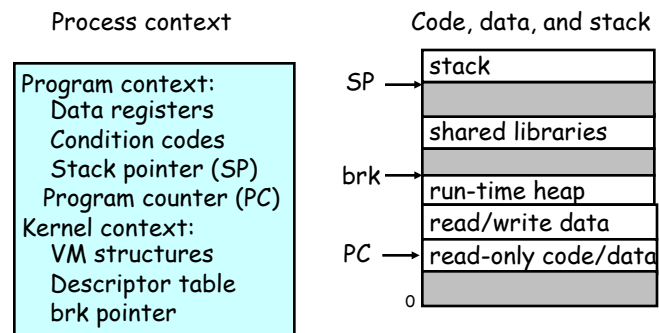
- ❑ Process = Address space + one thread of control
- ❑ Concurrent program = **multiple threads of control**
  - Multiple single-threaded processes
  - Multi-threaded process

CS 475

14

## Traditional View of a Process

- Process = process context + code, data, and stack

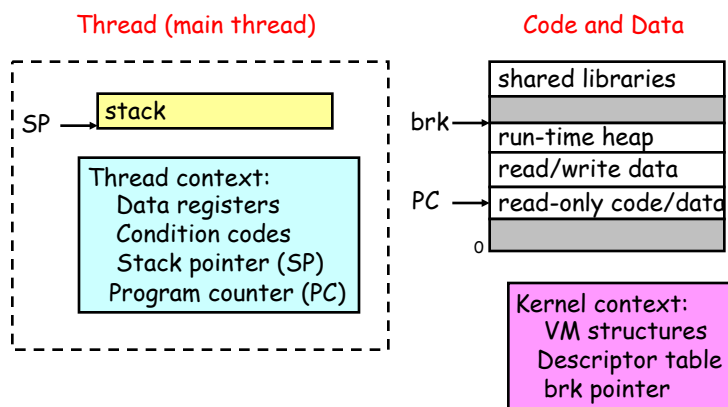


CS 475

15

## Alternate View of a Process

- Process = thread + code, data, and kernel context



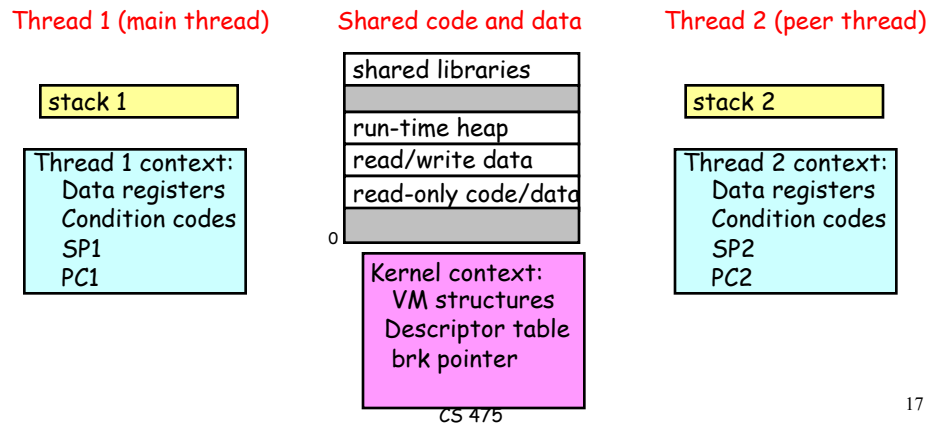
CS 475

16



## A Process With Multiple Threads

- ❑ Multiple threads can be associated with a process
  - Each thread has its own logical control flow (sequence of PC values)
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own thread id (TID)



17

## Threads: Motivation

- ❑ Traditional processes created and managed by the OS kernel
- ❑ Process creation expensive - fork system call in UNIX
- ❑ Context switching expensive
- ❑ Cooperating processes - no need for memory protection (separate address spaces)

CS 475

18

## Threads

- ❑ Execute in same address space
  - separate execution stack, share access to code and (global) data
- ❑ Smaller creation and context-switch time
- ❑ Can exploit fine-grain concurrency

CS 475

19

## Creating processes

- ❑ UNIX
  - **fork** system call
  - Used in conjunction with **exec** system call

CS 475

20

## fork: Creating new processes

❑ `int fork(void)`

- creates a new process (child process) that is identical to the calling process (parent process)
- returns 0 to the child process
- returns child's `pid` to the parent process

```
if (fork() == 0) {
    printf("hello from child\n");
} else {
    printf("hello from parent\n");
}
```

Fork is interesting  
(and often confusing)  
because it is called  
*once* but returns *twice*

## Creating and Using threads

❑ Pthreads Multi-threading Library

- API for thread creation and management
- `pthread_create`, `pthread_join`,  
`pthread_self`, `pthread_exit`,  
`pthread_detach`

❑ Java

- provides a `Runnable` interface and a `Thread` class as part of standard Java libraries
  - users program threads by implementing the `Runnable` interface or extending the `Thread` class

## Concurrent Systems

- ❑ Essential aspects of any concurrent system
  - **Execution context** - state of a concurrent entity
    - Processes: process context
    - Threads: thread context
  - **Scheduling** - deciding which context will run next
    - Processes: Operating System scheduler
    - Threads: Library thread scheduler (Pthreads), Java runtime
  - **Synchronization** - mechanisms that enable execution contexts to coordinate their use of shared resources
    - Semaphores, locks, monitors, condition variables
    - Provided at both operating system and library/language level

Distributed Software Systems

23

## Road Map

- ❑ Next two lectures: Processes & Signals in UNIX
  - Some of this will be a repetition of material discussed in CS 367
  - Assignment 1 (Shell Lab)
- ❑ Thread creation and management in Java and Pthreads (one lecture)
- ❑ Process & Thread synchronization mechanisms (two - three lectures)

CS 475

24