PURDUE
UNIVERSITY®

CS 25200: Systems Programming

Lecture 18: Syscall Wrap-up, Processes and Scheduling

Prof. Turkstra

Code Standard

- Lab 2 code standard analysis available soon
  - We won’t count this one :)

Office Hours

- Doubled up on two so far
- More to come after tonight’s meeting

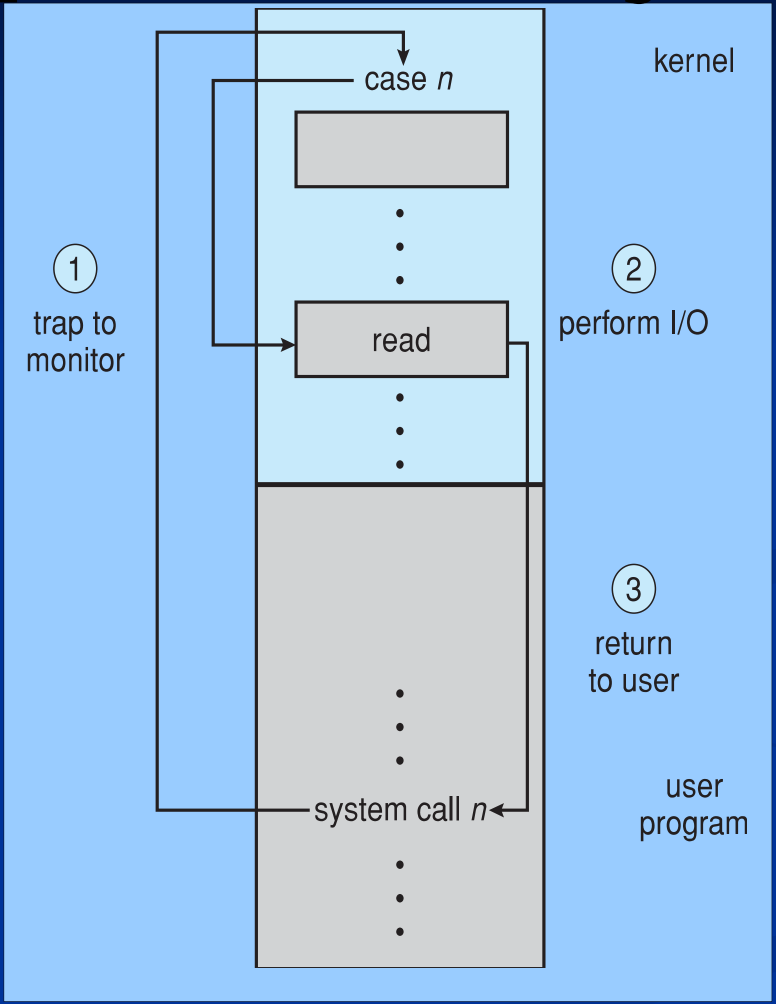
Checkpoint due Tonight

- 11:58pm
- We will not run it through valgrind
- Test cases

Lecture 18

- Syscall wrap-up
- Processes
- Scheduling

Example read system call



Synchronous write()

- Program executes the libc wrapper for write():
  write(fd, buff, n);
- libc places the arguments in the appropriate registers or stack locations
- libc then invokes syscall, which generates a software interrupt

- The OS interrupt handler checks the system call number and jumps to the appropriate location
- The handler verifies..
  - The fd is an open fd
  - Has the correct privileges (read/write/etc)
  - That [buff, buff+n-1] is a valid memory range
- Returns -1 and sets errno for failures

- OS determines the block(s) corresponding to the current file position by inspecting the inode
  - Also updates current file position
- OS sets up a DMA operation with the hard drive that takes the memory at buff, up to buff+n-1, and writes it to the appropriate block(s) address
- OS places current process in wait state

- OS switches to another process
- Disk completes write operation and generates a hardware interrupt
- OS jumps to appropriate ISR, writes the return value to rax and IRETs
- OS places process in the ready state
  - Available for scheduling

Security

- The checks that the kernel does on system call entry are critical
  - Never directly inspects user memory
- E.g., for open()…
  - Get file name and mode
  - Check if file exists
  - Verify permissions
  - Return fd

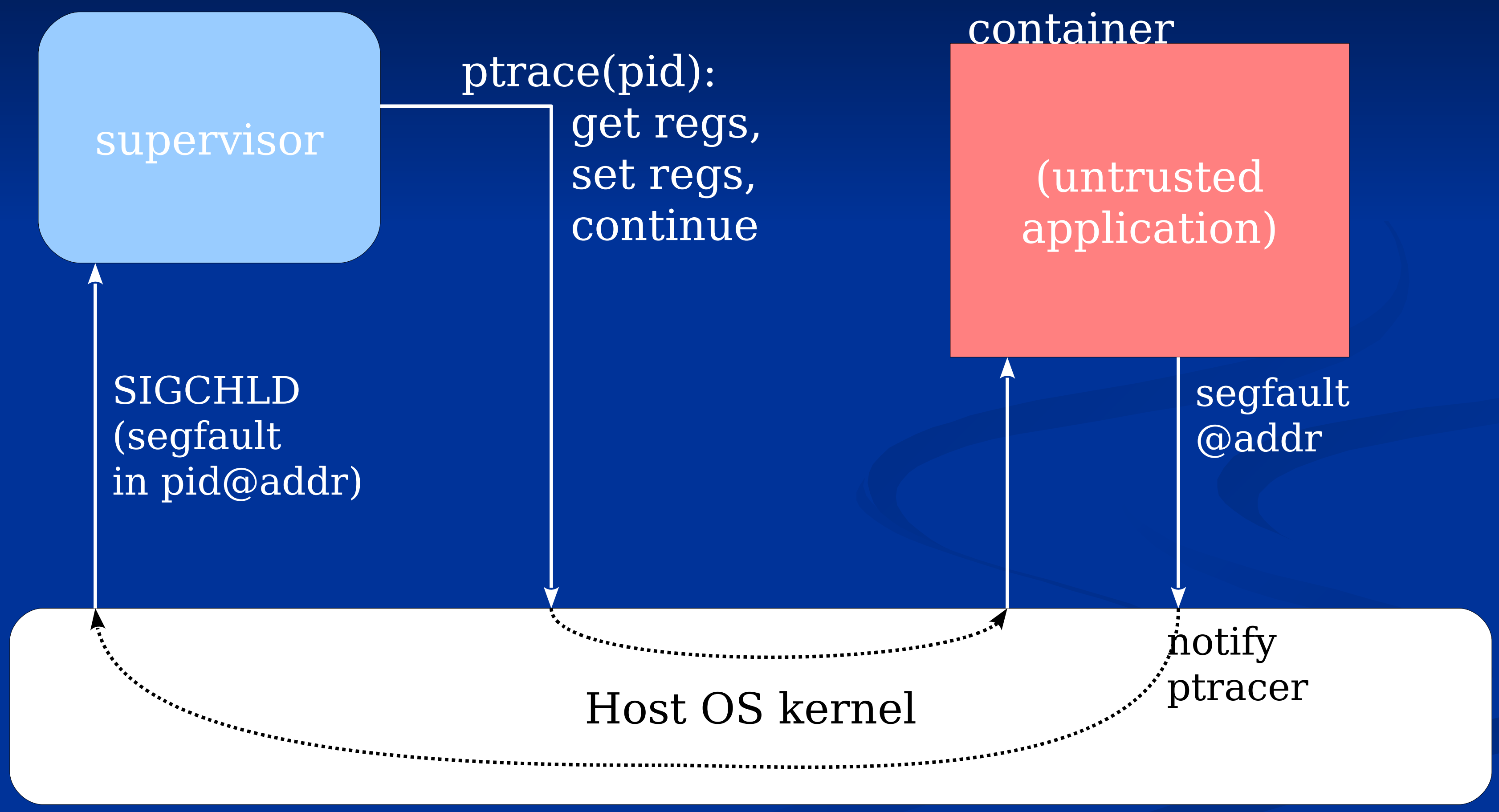
Errors

- Remember errno.h?
- System calls often return -1 and set errno
- man errno
- /usr/include/errno.h

strace

- Traces system calls and signals
  - Relies on the ptrace() system call
    - a “parent process” observes/controls another process
    - Can change child's core image and registers
    - Suspends child, wakes parent on *all* exceptional events

Handling a page fault



Interception slowdown

Type of container exception		Native (cycles)	Virtual (cycles)	Penalty
call getpid()	(min)	786.0	59442.0	75.6x
	(average)	1567.9	188051.6	119.9x
read fault		1329.5	90063.1	67.7x
write-after-read fault		3589.3	81826.3	22.8x
direct write double-fault		2924.4	170895.0	58.4x

Processes

- Programs may have multiple processes or instances running
  - E.g., multiple instances of Bash
- All processes have a parent
  - Except init, pid 1
- Remember ps?

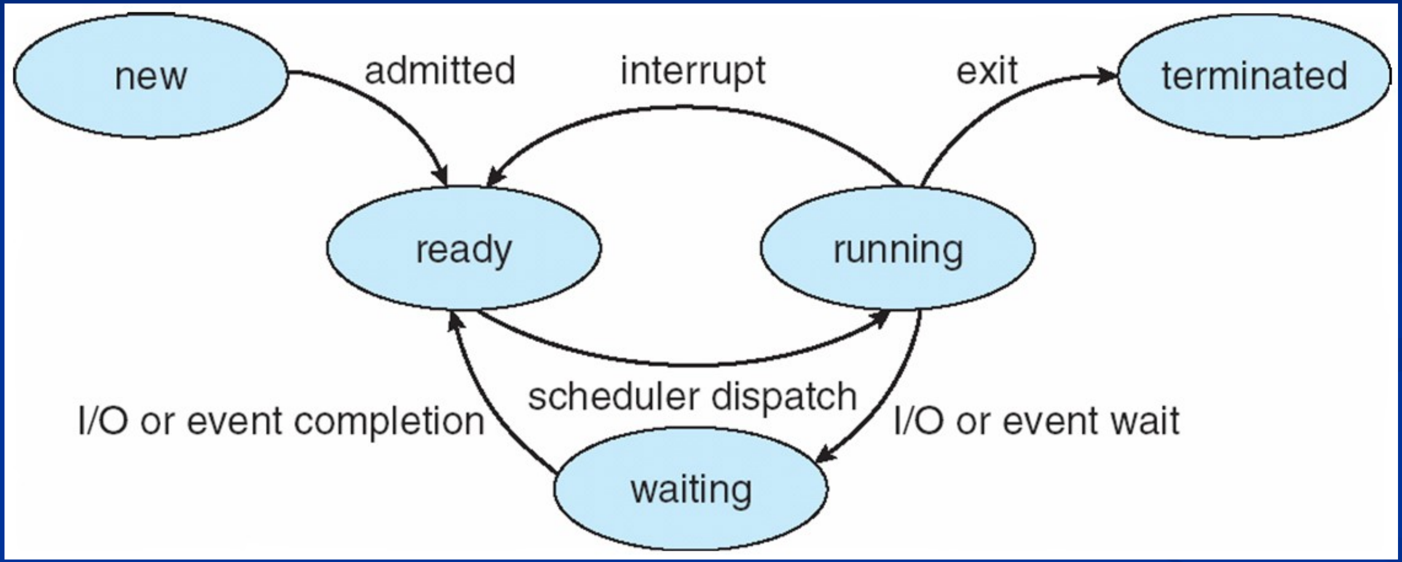
Processes

- top, ps (-e, -ax, -f, -u)

Process

- From an OS standpoint, a process includes...
  - Program counter
  - Registers
  - Memory mappings (page table)
  - File descriptor table
  - State
  - etc

Process states

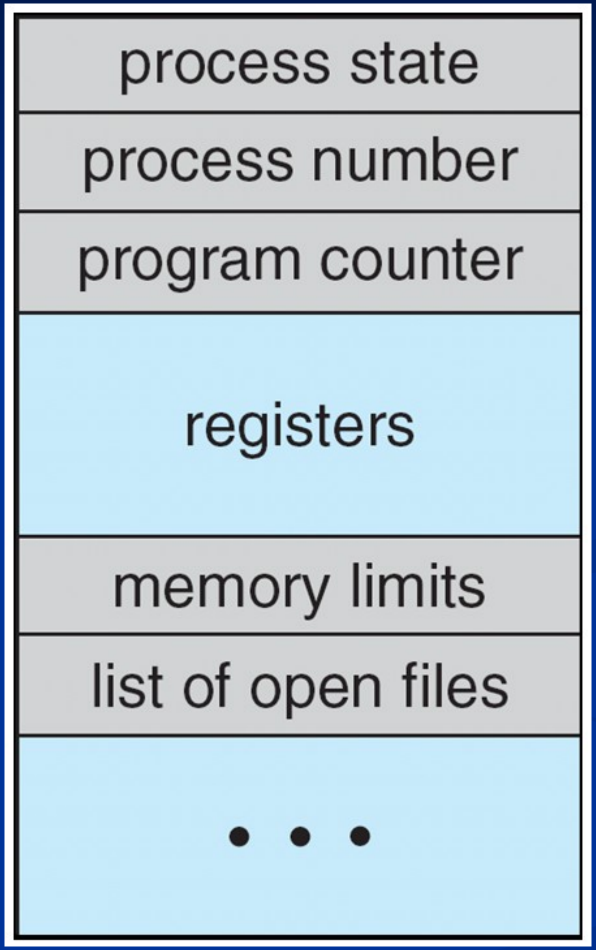


Process state

- New: process is being created
- Running: instructions are actually executing
- Waiting: waiting for an event to occur
  - Unable to run
- Ready: waiting to be assigned to a processor
  - Ready to run
- Terminated: process is done executing

Process control block (PCB)

- Process state – running, waiting, etc
- Program counter
- Registers
- Scheduling information – priorities, etc
- Memory mappings – page table
- Accounting information – CPU used, clock time, etc
- I/O status – I/O devices allocated to process
- Open files


process state
process number
program counter
registers
memory limits
list of open files
• • •

Process table

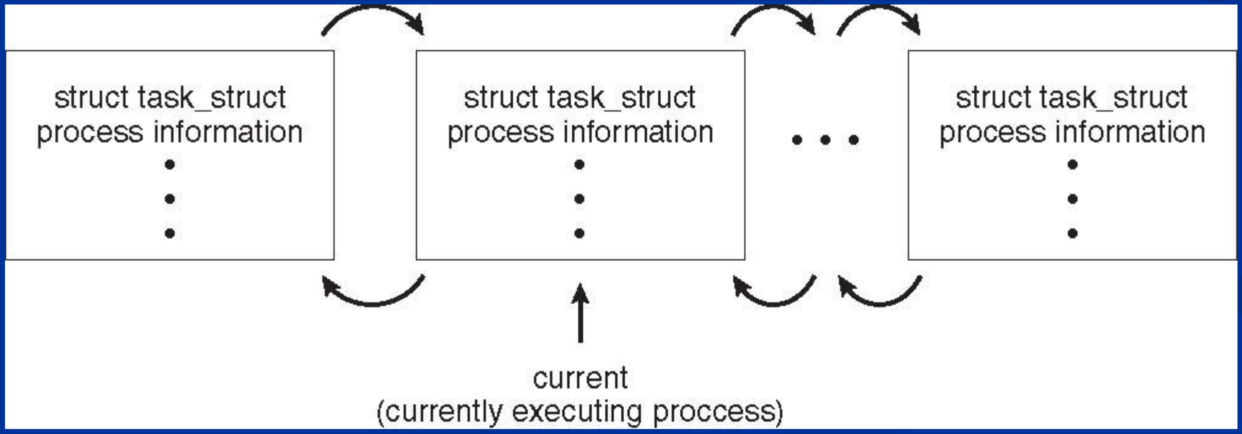
- A table of PCBs (process control blocks)
- One of the more important kernel data structures
- Maximum number of entries dictates maximum number of processes

Purdue Trivia

- The phrase “one brick higher” comes from the destruction of Heavilon Hall in 1894 – four days after construction was completed
  - Contained a groundbreaking locomotive testing plant
- President Smart proclaimed “We are looking this morning to the future, not the past… I tell you, …, that tower shall go up one brick higher!”
  - Actually nine bricks higher
- Current Heavilon Hall was built in 1959
  - Bells are in the Bell Tower (built 1995)
  - Clock is in the ME Gatewood Wing Atrium (2011)

Processes in Linux

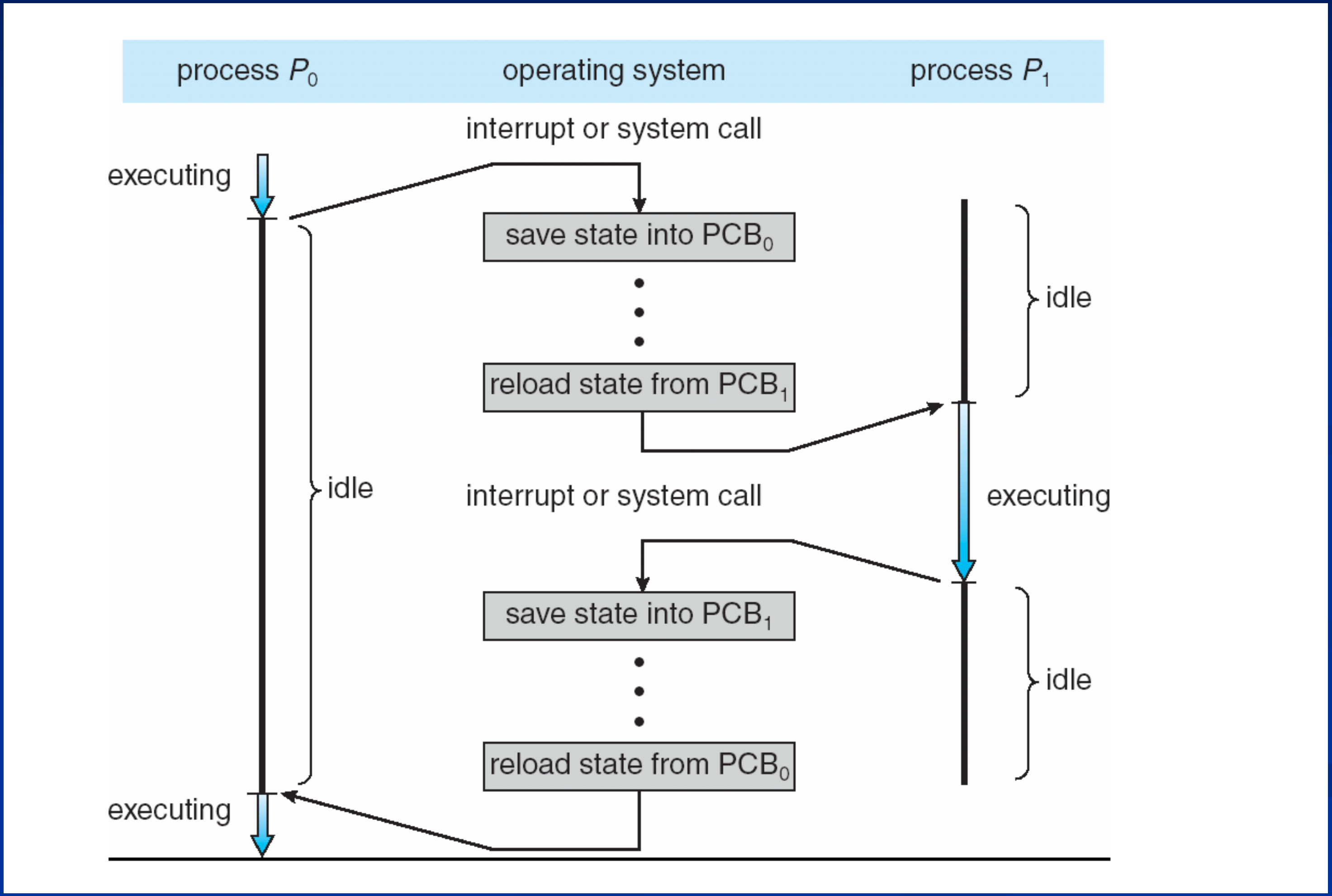
- struct task_struct
  - volatile long state
  - pid_t pid
  - struct list_head children
  - struct task_struct __rcu *parent
  - struct mm_struct *mm, *active_mm



Process tree

- ps faux

Context switch



Context switches

- happen when…
  - A process needs to wait on I/O
  - A process voluntarily yields
  - An interrupt occurs
  - The OS preempts the process

I/O vs. CPU bound

- I/O bound processes spend most of their time waiting
  - Mouse, keyboard, packet, etc
  - In ready/running state for short periods of time
- CPU bound processes spend most of their time ready or running
  - Scientific/numerical applications
  - Compilers, renderers, etc
- Most applications are I/O bound

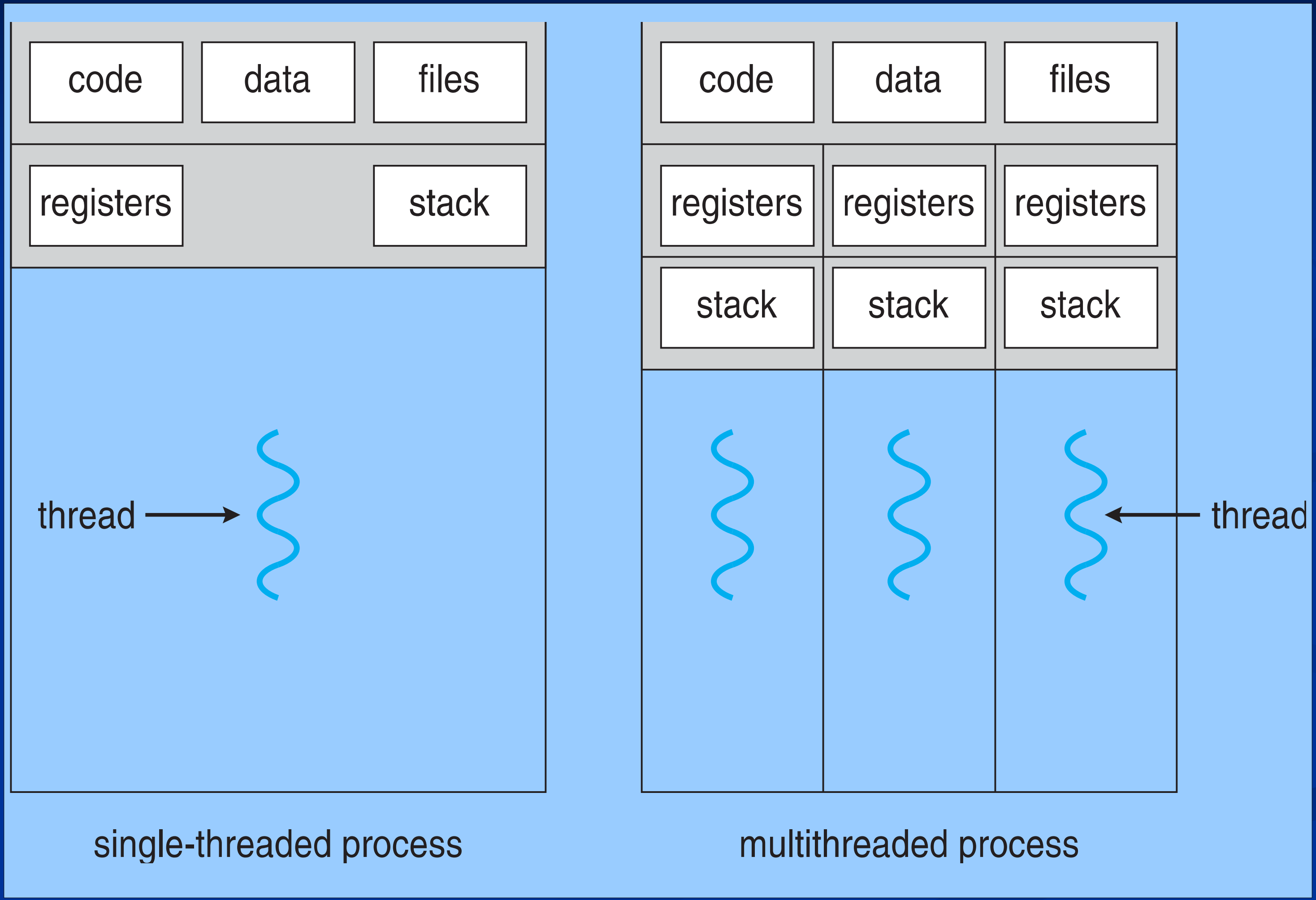
Symmetric multiprocessors

- Modern machines are usually SMPs
- Two or more identical processors sharing a common main memory
- Requires OS support
  - Originally Linux had the Big Kernel Lock (BKL)
    - Solitary, global lock that is held any time a processor enters kernel mode

Threads

- Process includes...
  - Address space (code, data, etc)
  - Resource container (OS resource, accounting)
  - A "thread" of control – PC, regs, stack
- Threads
  - Share some code and data (address space)
  - Same files, I/O channels, resource containers
  - Do not share thread of control

Threads



code
data
files
registers
stack
thread
code
data
files
registers
registers
registers
stack
stack
stack
thread
single-threaded process
multithreaded process

Threads

- Can have several threads in a single address space
- Threads are units of scheduling
- Processes are containers in which threads execute

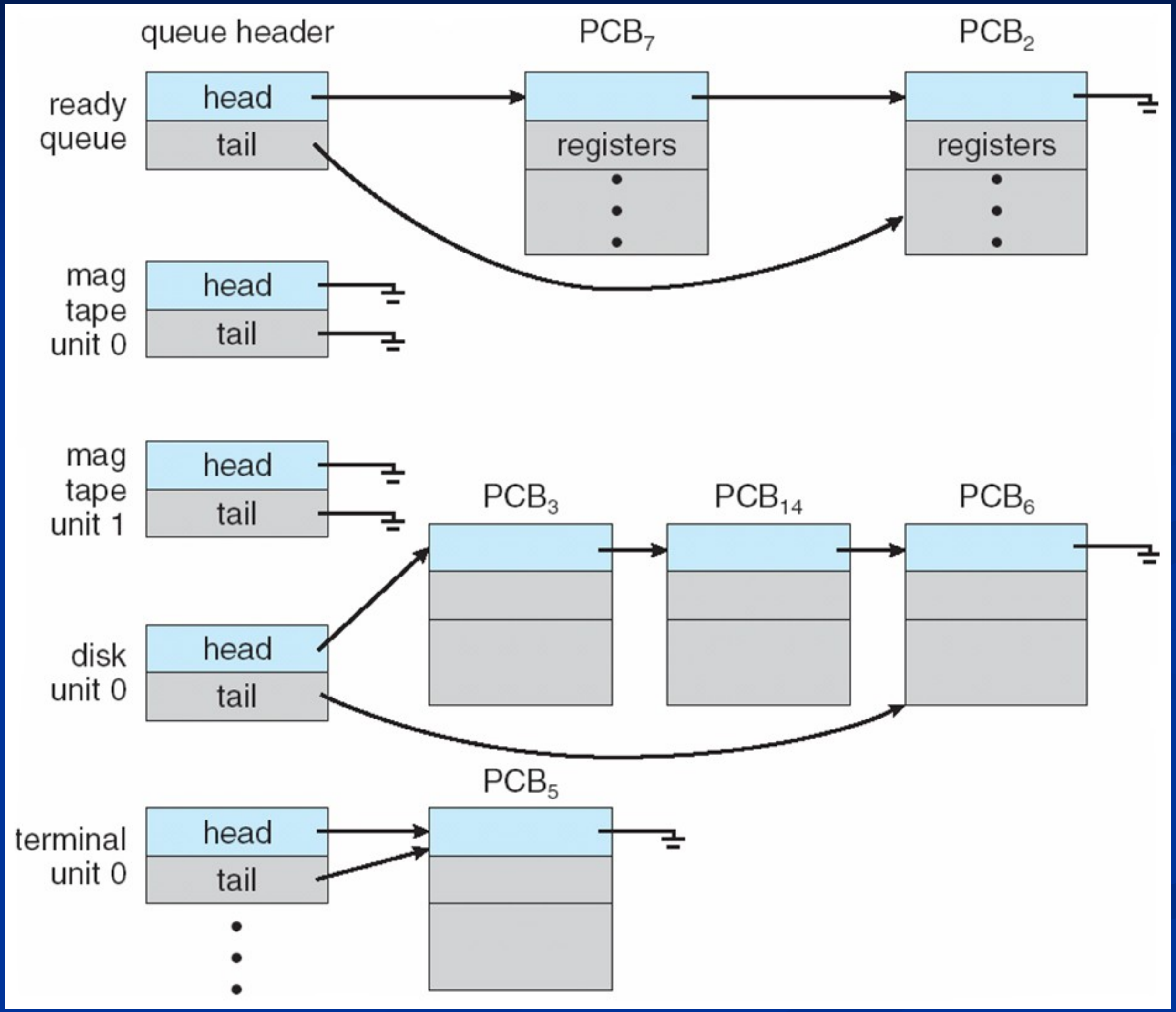
Threading

- Userland threads
  - POSIX Pthreads (IEEE 1003.1c)
  - Mach C-threads
  - Solaris threads
  - Windows threads
  - Java threads
- Kernel threads
  - Solaris LWP
  - Linux tasks (clone())
  - etc

Context switches

- TCB – Thread Control Block
  - Shared – parent process, execution time, memory, I/O resources
  - Private – PC, registers, stack, state information, pending/blocked signals
- TCB can be managed almost entirely in userland
  - Lower context switch overhead
- …or may rely on the kernel in some way

Process queues



Process scheduling

- From user standpoint, OS permits many processes executing simultaneously
- In reality, OS switches among processes rapidly to give the illusion of simultaneity

Scheduler

- Operating System subsystem that is responsible for determining which process(es) to run, for how long, and when
- Two types: non-preemptive and preemptive

Non-preemptive

- Context switches happen only when the running process waits or yields
- Also called cooperative multitasking
- Used in Windows 3.1 and initial versions of MacOS

Preemptive

- Context switches can be forced
  - Usually after a fixed period of time, called a quantum
  - E.g., every 1/100sec
- Rely a timer interrupt that invokes the OS scheduler
  - Often the process that has been in the ready state the longest will execute next
- Implemented in *NIX, Windows 95 and above, etc

Tradeoffs

- Non-preemptive
  - More (user) control over how the CPU is used
  - Simple
- Preemptive
  - More robust
  - Enforced fairness

Questions?