Putting It All Together

CS439: Principles of Computer Systems

May 6, 2015

Last Time

Something Different

Today's Agenda

Take a step back, look at how the pieces fit together.

From Architecture to OS to Application and Back

Hardware	**Example OS Services**	**User Abstraction**
Processor	Process management, Scheduling, Traps, Protections, Billing, Synchronization	Process
Memory	Management, Protection, Virtual memory	Address space
I/O devices	Concurrency with CPU, Interrupt handling	Terminal, Mouse, Printer, (System Calls)
Disk	File System (Management, Persistence)	Files
Distributed systems	Network security, Distributed file system	RPC system calls, Transparent file sharing

From Architecture to OS to Application and Back
Plain Text

- Processor
  - OS Services: Process management, Scheduling, Traps, Protections, Billing, Synchronization
  - User Abstraction: Process
- Memory
  - OS Services: management, protection, virtual memory
  - user abstraction: address space
- I/O Devices
  - OS Services: concurrently with CPU, Interrupt handling
  - User Abstraction: terminal, mouse, printer, (system calls)
- Disk
  - OS Services: file system (management, persistence)
  - User Abstraction: Files
- Distributed systems
  - OS Services: network security, distributed file system
  - User Abstraction: RPC system calls, transparent file sharing

From Architecture to OS to Application and Back

OS Service	**Hardware Support**
Protection	Kernel / User mode Protected Instruction MMU
Interrupts	Interrupt Vectors
System calls	Trap instructions and trap vectors
I/O	Interrupts or Memory-Mapping
Scheduling, billing	Timer
Synchronization	Atomic instructions
Virtual Memory	Translation look-aside buffers Register containing base of page table (PTBR)

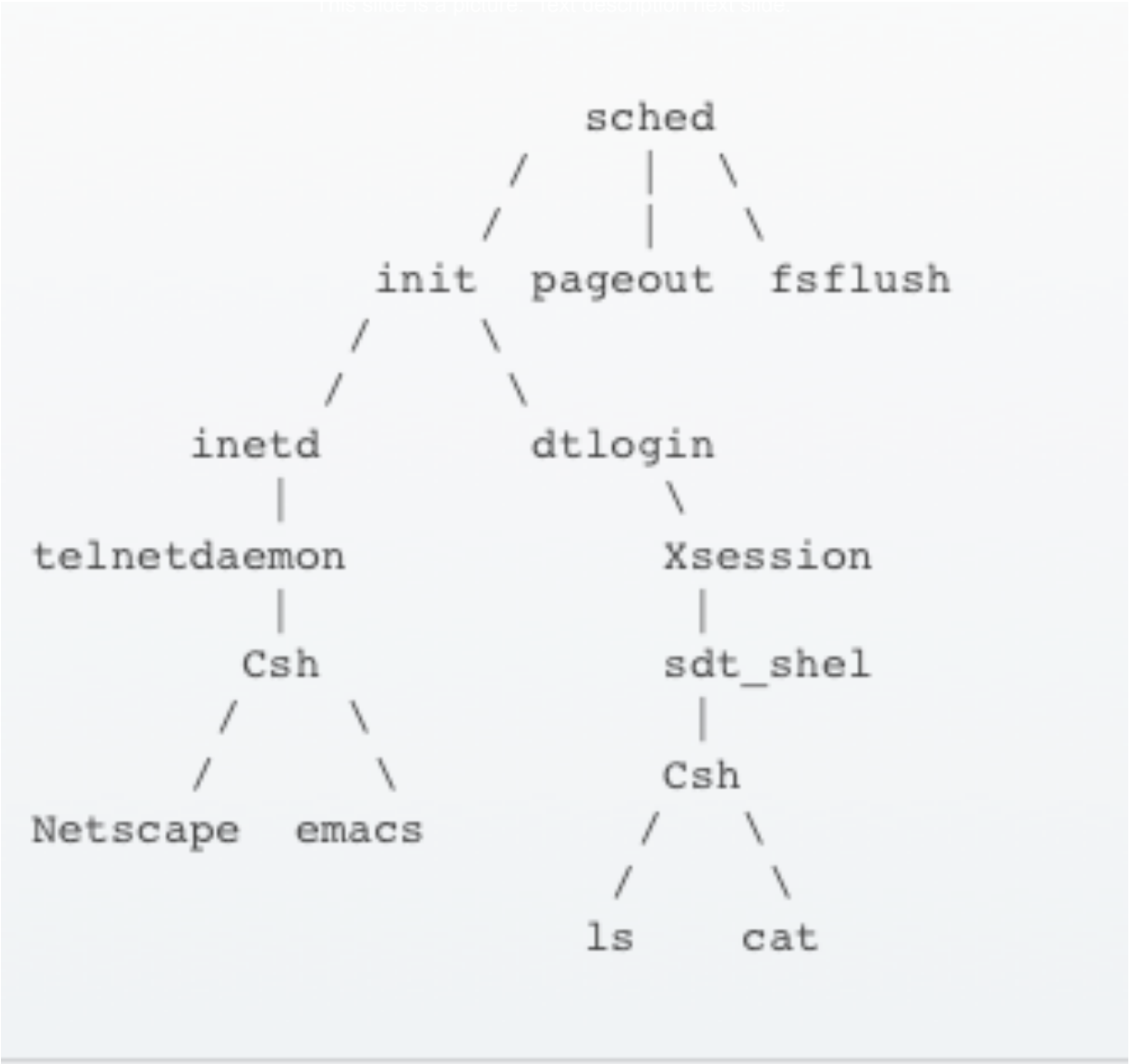
From Architecture to OS to Application and Back Plain Text

- Protection
  - Hardware support: kernel/user mode, protected instructions, MMU
- Interrupts
  - Hardware support: interrupt vectors
- System calls
  - Hardware support: trap instructions and trap vectors
- I/O
  - Hardware support: interrupts or memory-mapping
- Scheduling and billing
  - Hardware support: timer
- Synchronization
  - Hardware support: atomic instructions
- Virtual memory
  - Hardware support: translation lookaside buffers, register point to base of page table (PTBR)

An Operating System in Action

- CPU loads boot program from ROM (e.g. BIOS in PCs)
- Boot program:
  - Examines/checks machine configuration (number of CPUs, how much memory, number & type of hardware devices, etc.)
  - Builds a configuration structure describing the hardware
  - Loads the operating system, and gives it the configuration structure
- Operating system initialization:
  - Initialize kernel data structures
  - Initialize the state of all hardware devices
  - Creates a number of processes to start operation

Initial Processes



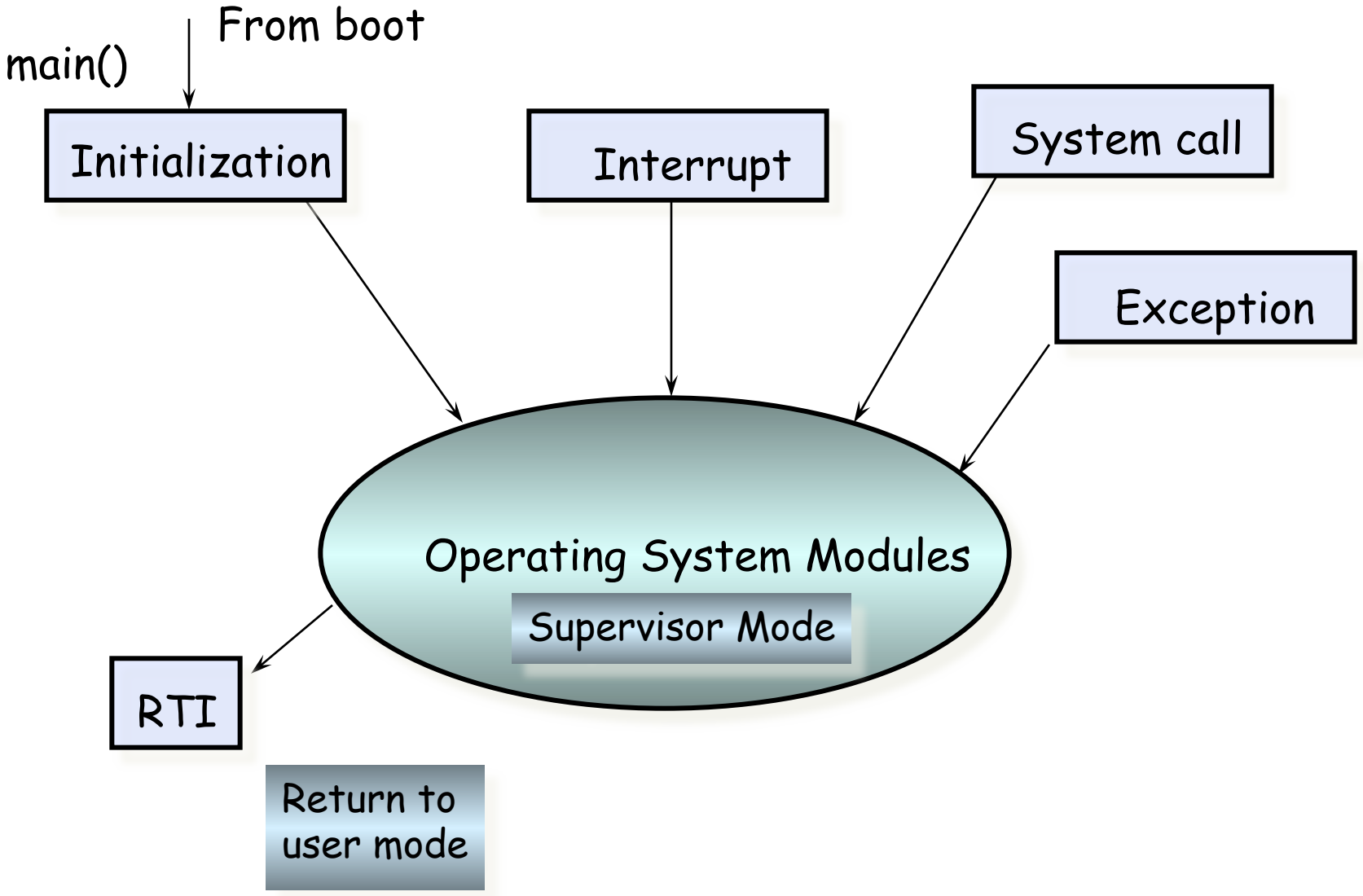
Initial Processes: Text Description

- Initial Process: sched
- sched has 3 children: init, pageout and fsflush
- init has 2 children: inetd and dtlogin
- inetd has child telnetdaemon has child Csh
  - Csh has 2 children: Netscape and emacs
- dtlogin has child Xsession which has child sdt_shel has child Csh (different instance from inetd's child)
  - Csh has 2 children: ls and cat

An Operating System in Action

- After basic processes have started, the OS runs user programs or enters the idle loop
- After basic processes have started, the OS runs user programs, if available, otherwise enters the idle loop
  - In the idle loop:
    - OS executes an infinite loop (UNIX)
    - OS performs some system management & profiling
    - OS halts the processor and enter in low-power mode (notebooks)
    - OS returns from idle on an interrupt
- OS code is executed after (you know this!):
  - Interrupts from hardware devices
  - Exceptions from user programs
  - System calls from user programs
- Two modes of execution
  - User mode: Restricted execution mode (applications)
  - Supervisor mode: Unrestricted access to everything (OS)

Control Flow in an OS



Control Flow in an OS: Text Description

- There are multiple ways to enter the operating system modules (supervisor mode)
  - Interrupt
  - System call
  - Exception
- You also enter supervisor mode during the initialization of the system from boot time
- RTI, the return from interrupt call, returns to user mode

So… The Unix Shell

- When you log in to a machine running Unix, the OS creates a shell process for you to use
- Every command you type into the shell is a *child* of your shell process
  - For example, if you type “ls”, the OS creates a new process and then execs ls

iClicker Question

What creates a process?

A. fork()
B. exec()
C. both

iClicker Question

When creating a new process, fork() creates an exact copy of the old process except the:

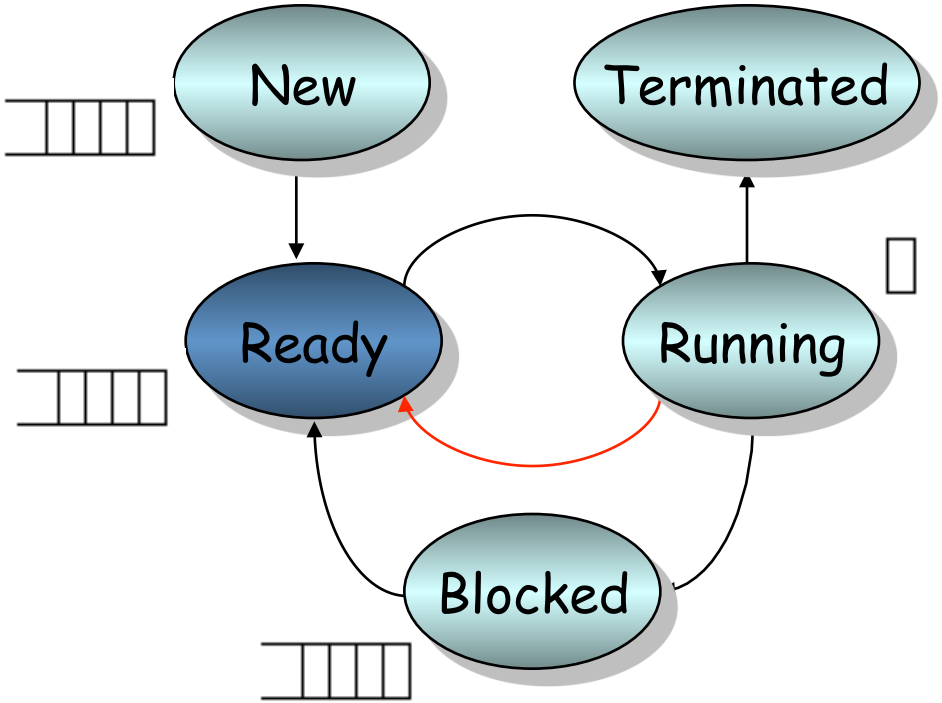
A. Stack
B. Program counter
C. PID
D. Heap

Processes: High Level

- Enter system
  - Initialization includes creating the PCB
- Enter Ready queue
- Scheduled to run
  - Address space is demand paged into memory
  - If memory is full, page replacement algorithm executes
- Stops running when:
  - Finishes, OS deallocates
  - Quantum over, alarm causes interrupt, context switch occurs (if pre-emptive scheduler)
  - On I/O, blocks, context switch occurs

Process Life Cycle

All of the processes the OS is currently managing reside in exactly one of these state queues.



Process Life Cycle: Text Description

- All of the processes the OS is currently managing reside in exactly one of these state queues
  - States: new, ready, running, blocked, and terminated
- Possible state changes
  - new to ready
  - ready to running or running to ready
  - running to blocked
  - blocked to ready
  - running to terminated
- Note: there is only ever one running process so the queue for that state is of size one
  - We are assuming uniprocessors

iClicker Question

When a process is waiting for I/O, what is its scheduling state?

A. Ready

B. Running

C. Blocked

D. Zombie

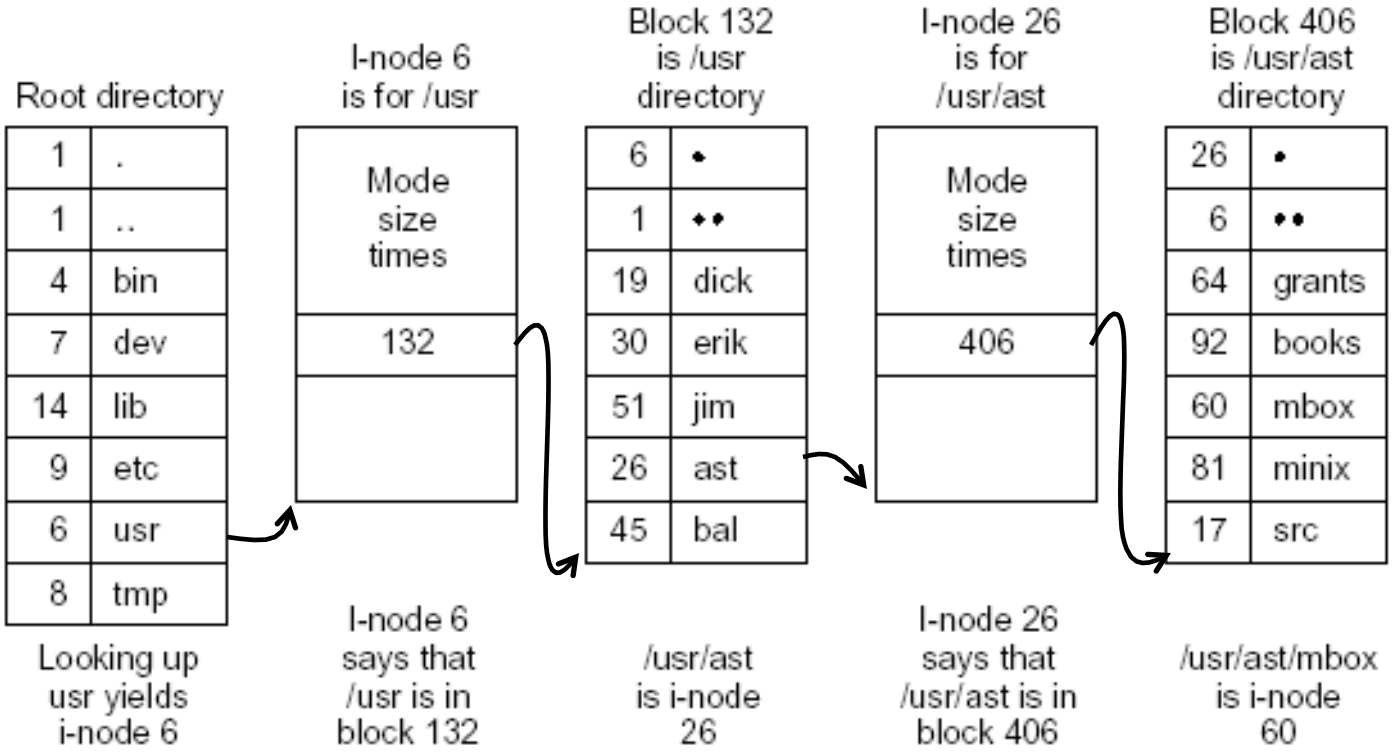
E. Exited

I/O: Sets OS in Motion

- I/O requested through system call, control returns to OS
- Let's assume a file system read in a Linux system…

- OS begins by checking its file cache to see if the inode for that file is cached, if it isn't, it will need many disk accesses…

Reading a File



The steps in looking up */usr/ast/mbox*.

- Root directory is a file itself and its inode is at a fixed location on
  disk.
- inodes are all clustered together in a fixed region on disk

Reading a File: Text Description

- The steps in looking up /usr/ast/mbox on the disk
  1. Retrieve root directory inode from fixed location on disk
  2. Retrieve root directory data block from location given in inode.
  3. Look up usr in root directory data block; its information is in inode 6
  4. Go to inode 6 and determine which block holds the data for /usr
     - This is block 132
  5. Get block 132, the /usr directory data, and lookup ast; its information is in inode 26
  6. Go to inode 26 and determine which block holds the data for /usr/ast
     - This is block 406
  7. Get block 406, the /usr/ast data, and lookup mbox; its information is in inode 60
     - Woot! We have found our email!
- Also, note:
  - Root directory is a file itself and its inode is at a fixed location on disk.
  - inodes are all clustered together in a fixed region on disk

I/O: OS Schedules

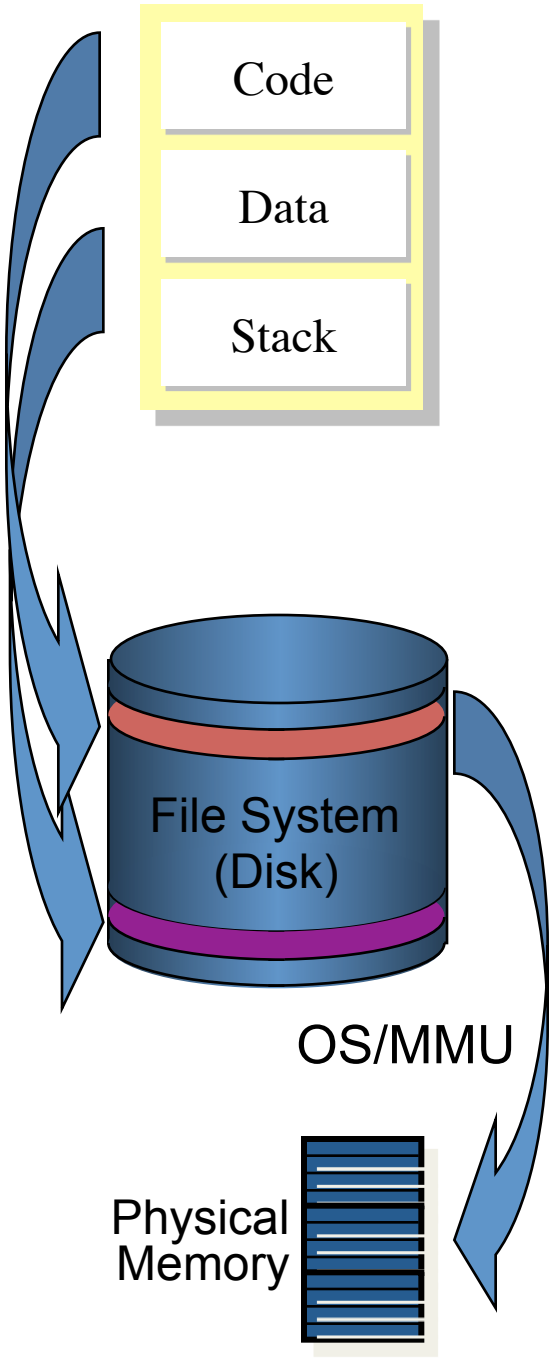
- OS begins requesting information from the disk
- Adds reads to disk queue
- When read is scheduled, issues command to disk controller
  - 3 methods of communication
- As inodes are retrieved from disk, OS checks permissions and owner, locates data blocks
- Eventually, data blocks are retrieved and copied into memory
  - Copied into free page frame
  - What if no page frames are free? Page replacement algorithm!

Once I/O is requested… Context Switch

- Requesting process blocks
- Operating system chooses a process to run
- Loads its register contents
- May flush the TLB
  - Some TLBs may have PIDs in them
- Updates MMU with page table of that process

What is a Process's Context?

- Registers, etc., which are stored in PCB are part of a process's context
- A process's VAS is part of its context
  - Contains its code, data, and stack
- OS determines which portions of a process's VAS are mapped in memory at any one time
- Code pages are stored in a user's file on disk
  - Some are currently residing in memory; most are not
- Data and stack pages are also stored in a file
  - Although this file is typically not visible to users
  - File only exists while a program is executing



What is a Process's Context? Plain Text

- Registers, etc., which are stored in PCB, are part of a process's context
- A process's VAS is part of its context
  - Contains its code, data, and stack
- OS determines which portions of a process's VAS are mapped in memory at any one time
- Code pages are stored in a user's file on disk
  - Some are currently residing in memory; most are not
- Data and stack pages are also stored in a file
  - Although this file is typically not visible to users
  - File only exists while a program is executing

When data from read is ready…

- If not polling, interrupt is issued
- OS regains control, checks register to determine which interrupt service routine to run
- Handles interrupt
- In the case of our example read, copies data to requesting process's memory (if necessary)
- Runs scheduler (maybe)
- Returns control to user process through RTI instruction

What If Too Many Processes are in the System?

- Swap some out to disk!
- Save a process's state to disk and remove it from memory
- Frees up memory
- Swapping was also used to eliminate fragmentation from memory before paging

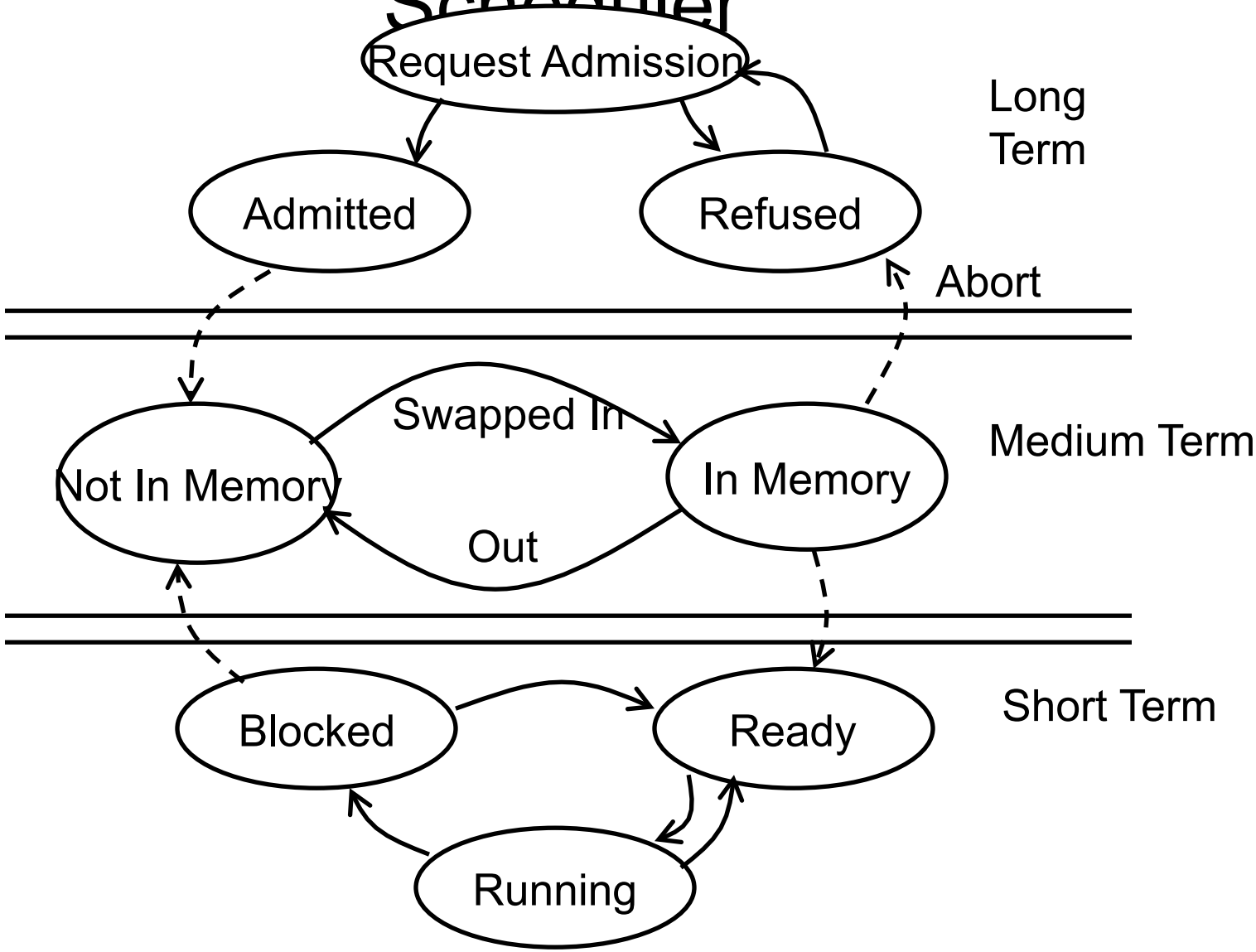
When does the OS swap out processes?

Process Scheduling

Scheduling Goals:

- Long Term: Prevent overloading (admission control)
- Medium Term: Reasonable mix of I/O and CPU bound jobs, thrashing control (Swapping!)
- Short Term: Efficient use of CPU (scheduling algorithms we studied)

States of a Task in an Hierarchical Scheduler



States of a Task in an Hierarchical Scheduler: Text Description

- 3 different types of scheduling: long term, medium term and short term
- States for long term: request admission, admitted and refused
  - Possible transitions:
    - request admission to refused and refused to request admission
    - request admission to admitted
- States for medium term: not in memory and in memory
  - Possible transitions:
    - non in memory to in memory (swapped in)
    - in memory to not in memory (swapped out)
- States for short term: blocked, ready and running
  - Possible transitions:
    - blocked to ready
    - ready to running and running to ready
    - running to blocked
- Transitions between types of scheduling:
  - admitted (long) to not in memory (medium)
  - in memory (medium) to refused (long) - is an abort
  - blocked (short) to not in memory (medium)
  - in memory (medium) to ready (short)

Where are the threads?

- User-level: a thread the OS does not know about
  - Kernel only manages the process
  - Programmer uses a *thread library* to manage threads (create, delete, synchronize, and schedule)
- Kernel-level: a thread that the OS knows about
  - Kernel manages and schedules

iClicker Question

In the same process, threads each have their own:

A. Stack
B. Registers
C. Address Space

iClicker Question

Every monitor function should begin with what command?

A. Wait()

B. Signal()

C. Lock->Acquire()

D. Lock->Release()

E. Broadcast()

Where do we go from here?

- Memory is getting cheaper
- Network is faster than the disk
- More and more parallelism
  - GPUs
  - Multi-core
  - Distributed systems

Other Courses

- Operating Systems
  - Driver writing! and other topics
- Programming for Performance
- Distributed Computing
  - Three-phase commit
  - Bayou
- Networking
- Security

iClicker Questions

Given:

- a memory size of 256 addressable words
- a page table indexing 8 pages
- a page size of 32 words
- 8 logical segments

- How many bits is a physical address?
  A. 6 B. 8 C. 11 D. 13
- How many bits is a virtual address?
  A. 6 B. 8 C. 11 D. 13
- How many segment table entries do we need?
  A. 6 B. 8 C. 11 D. 13

Summary

- An OS is just a program:
  - It has a main() function, which gets called only once (during boot)
  - Like any program, it consumes resources (such as memory), can do silly things (like generating an exception), etc.
- But it is a very strange program:
  - It is “entered” from different locations in response to external events
  - It does not have a single thread of control, it can be invoked simultaneously by two different events (e.g. system call & an interrupt)
  - It is not supposed to terminate
  - It can execute any instruction in the machine

Summary

- We've learned many design principles from the OS:
  - To speed things up, add…
  - You can solve any problem by adding a …
- There is always a trade off between time and space
- Locality matters
  - Spatial and temporal!

Announcements

- Final Exam is Thursday, 5/14, at 7pm in UTC 2.102A
  - Final exam study guide will be posted by Friday
- Project 4 due Friday at 11:59p
  - No slip days!
- No discussion sections on Friday
- Consult Piazza for additional office hours
  - Friday discussion sections (there now)
  - Finals week (coming soon)
- Group member evaluations are coming (and required)
  - Slightly more detailed than usual
  - Includes every group member you have had
  - Due by 5/9 at 12p