

Processes

- What is a process?
- How are they represented inside the OS?
- How are they created?
- What are the possible execution states of a process?
 - How does the system move between them?
- How is the CPU scheduled across processes?

What is a Process?

- A process is the OS's abstraction for a running piece of code.
 - Remember play vs. script analogy from class!
- Container for a lot of state:
 - Address space
 - Code for running program
 - Data for running program
 - CPU registers
 - Pointer to stack
 - Program counter
 - Etc. . .
- Named by a *process ID* (PID)
 - Usually just an integer!

How are They Represented by the OS?

- The data structure in the OS that represents a process is called a Process Control Block (PCB)
- PCB stores all the process state when the process isn't running.
 - Contents of registers
 - Program counter
 - What else?
- Example: in the Linux OS, this is defined in `task_struct` in `include/linux/sched.h`
 - Over 95 fields!

How is a Process Created?

- One process can make another process
 - In UNIX: `fork()` system call.
 - Returns twice; once into parent and once into child.
 - Creator is the parent, created is the child
- The created process is an **exact** clone of the parent.
 - PCB, memory, etc.
 - Another program is then loaded into its memory space.
 - In UNIX: `exec()` system call.
 - Is this efficient? Could we do better?
- Parent can wait for child to finish or run in parallel

What are the States of a Process?

- Each process has an 'execution state', which describes what it is currently doing.
 - Ready: can be scheduled to do work at any time.
 - Running: currently controls the CPU
 - How many processes can be running simultaneously?
 - Waiting: blocked while waiting for an event.
 - IO completion, permission to enter critical area, etc.
- A process moves from state to state as it executes.
 - What does the process state diagram look like?

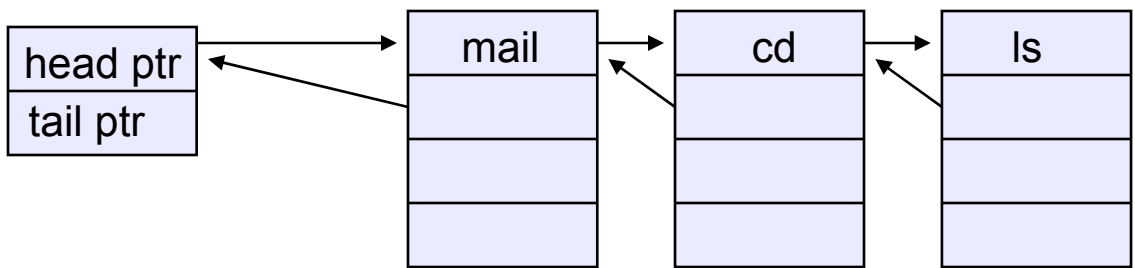
How are Processes Scheduled?

- The act of switching the CPU from one process to another is called a *Context Switch*.
 - Stop running the previous process
 - Save its state
 - Load the state of the new process
 - Continue
- The part of the OS that decides which process to run is called the Scheduler.
 - When is it run?
- The OS maintains a set of queues that represent the state of all processes on the system.
 - Linked list of PCBs.
 - A process moves from queue to queue as it changes state.
 - How many queues do we need?

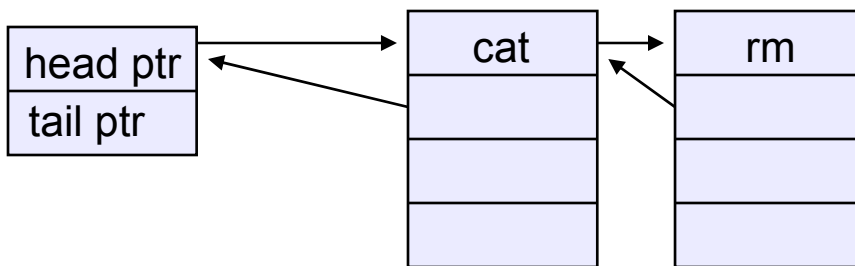
Process Queues

- Example:

Ready queue header



Wait queue header



- There may be several wait queues of different priority.
- How might a process 'give up' the cpu and switch to another queue?
 - Hint: there is more than one way!

Interrupts and Traps

- Traps are caused by software executing instructions
 - e.g. the x86 'int' instruction
 - e.g. a page fault, write to a read-only page
 - an expected exception is a “trap”, unexpected is a “fault”
- Interrupts are caused by hardware devices
 - e.g. device finishes I/O
 - e.g. timer fires
- A hardware interrupt is caused when a device sends an interrupt signal on the bus.
 - Example: hitting a key on the keyboard.
 - In memory, a **vector table** contains list of addresses of kernel routines to handle various interrupt types
 - Who populates the vector table, and when?
 - CPU switches to address indicated by vector specified by interrupt signal

More on Interrupts and Traps

- Interrupts are basis for asynchronous I/O
 - Device performs an operation asynch to CPU
 - Device sends an interrupt signal on bus when done
- How can the OS prevent runaway user programs from hogging the CPU (infinite loops?)
 - Use a hardware timer that generates a periodic interrupt
 - Before it transfers to a user program, the scheduler loads the timer with a time to interrupt
 - “quantum”: how big should it be set?
 - When timer fires, an interrupt transfers control back to OS
 - at which point OS must decide which program to schedule next