CPU Scheduling

CS439: Principles of Computer Systems

February 2, 2015

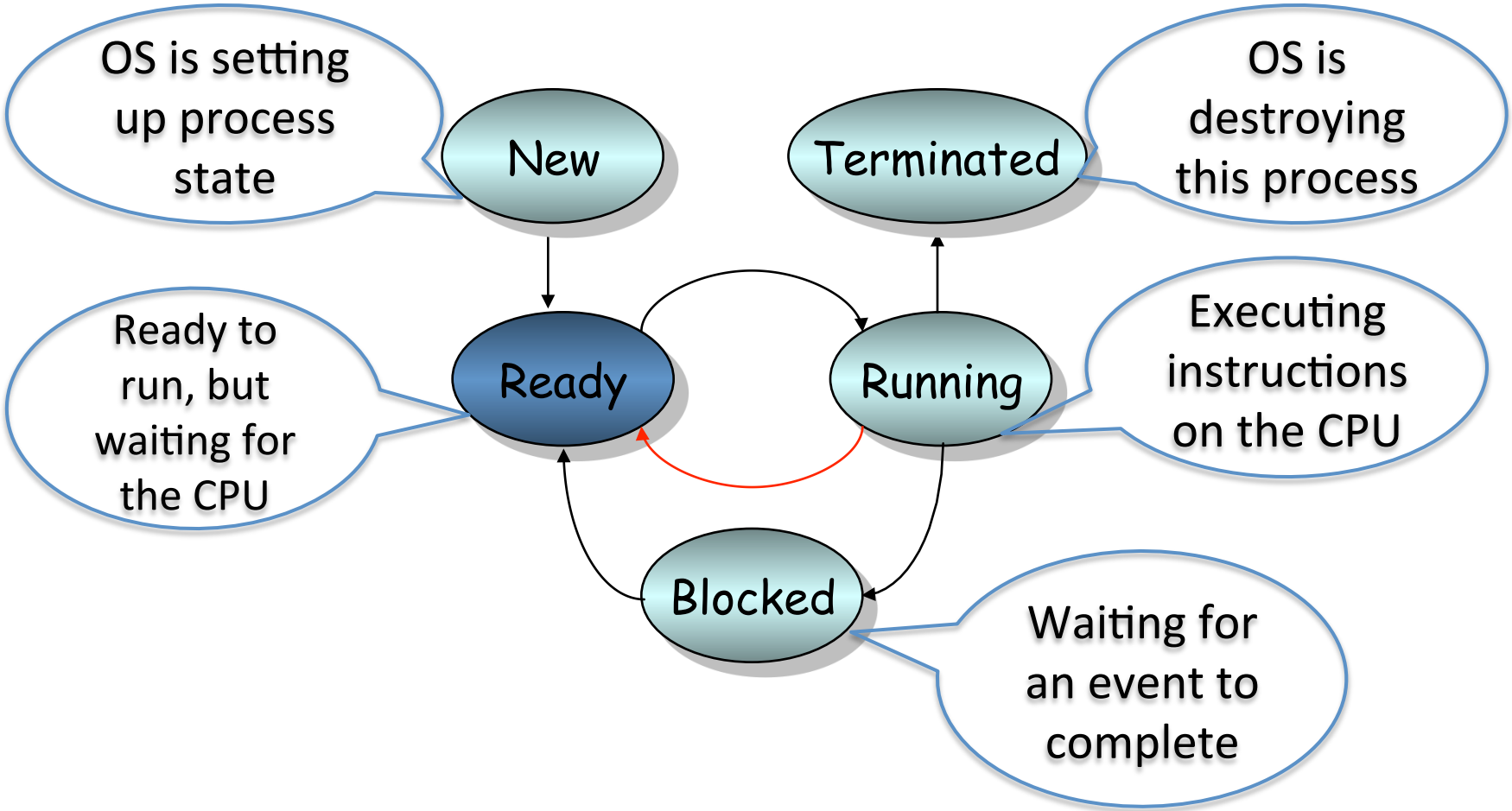
Last Time

A process is a unit of execution

- Defines an address space
- An abstraction for protection
- Processes are represented as Process Control Blocks in the OS
- At any time, a process is either New, Ready, Blocked, Running, or Terminated
- Processes are created and managed through system calls
  - system calls exist for other things, too

Recall: Process Life Cycle

Processes are always either *running*, *ready to run* or *blocked waiting for an event* to occur



Today's Agenda

- Boot Sequence
- CPU scheduling
  - FCFS, Round Robin, Shortest Job First, Multilevel Feedback Queues

Boot Sequence

Booting an OS kernel

- 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 OS kernel 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 (e.g. getty in UNIX, the Windowing system in NT, etc.)

O.S. in Action (Cont'd)

- 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

CPU Scheduling

Scheduling Processes

The OS uses *multiprogramming*, or *concurrency, (*one process on the CPU running and one or more doing I/O) to increase system utilization and throughput by overlapping I/O and CPU activities.

Questions to Consider

- Long Term Scheduling
  - How does the OS determine the degree of multiprogramming, i.e., the number of jobs executing at once in the primary memory?
- Short Term Scheduling:
  - How does (or should) the OS select a process from the ready queue to execute?

Short Term Scheduling

The *scheduler* schedules the next process on the CPU

- it is the *mechanism*
- it implements a *policy*

The kernel runs the scheduler when:

- a process switches from running to blocking
- a process is created or terminated
- an interrupt occurs
  - I/O device finishes
  - Periodic timer

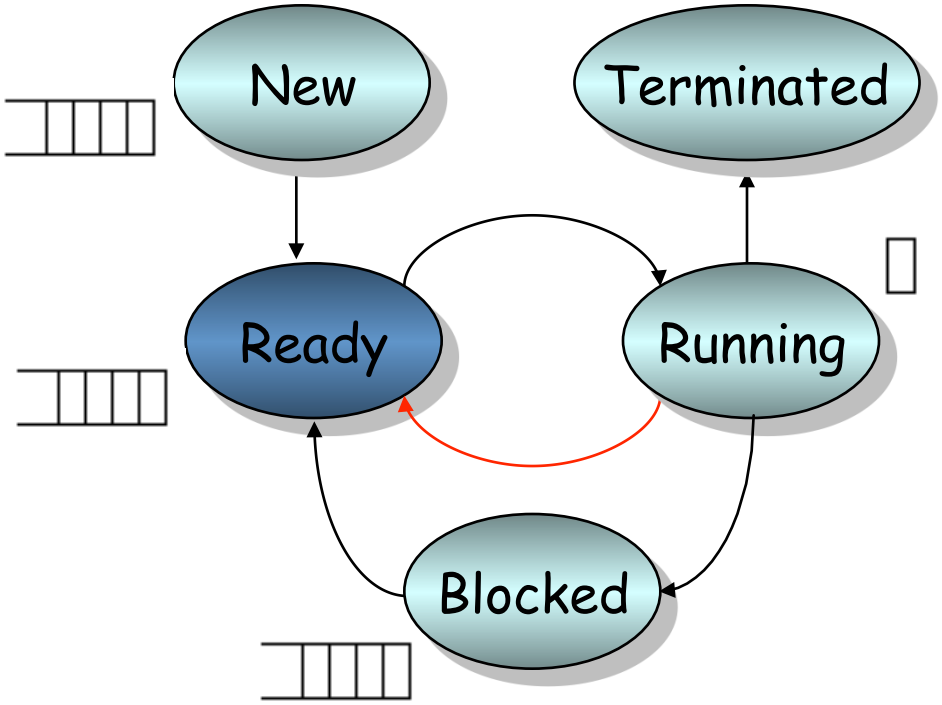
Scheduling algorithms are divided into categories based on their response to interrupts.

Scheduling Policies: Two Big Categories

- Non-preemptive
  - Scheduler runs when process blocks or terminates---not on hardware interrupts
- Preemptive
  - OS makes scheduling decisions during interrupts, mostly timer, but also system calls and other hardware device interrupts
  - Requires interrupt to give control back to the OS
  - Most OSes today have this

Process Life Cycle

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



Criteria for Comparing Scheduling Policies

- **Throughput**: number of processes completing in a unit of time
- **CPU Utilization**: percentage of time that the CPU is busy
- **Turnaround time:** length of time to run a process from initialization to termination, including all the waiting time
- **Response time**: time between issuing a command and getting a result
- **Waiting time**: total amount of time that a process is in the ready queue.

Scheduling Policies

- Ideal CPU scheduler
  - Maximizes CPU utilization and throughput
  - Minimizes turnaround time, waiting time, and response time
  - Requires knowledge of process behavior we typically don't have
- Real CPU schedulers implement particular policy
  - Minimize response time
  - Minimize variance of average response time
    - predictability may be more important than a low average with a high variance.
  - Maximize throughput
    - minimize overhead (OS overhead, context switching)
    - efficient use of system resources (CPU, I/O devices)
  - Minimize waiting time
    - be *fair* by ensuring each process waits similar amounts of time
    - often increases average response time.

Scheduling Policies: Context Switches

- Changing from one process to another is known as a *context switch*
- Steps:
  - Process is running
  - Process blocks or is interrupted
  - Mode switch to kernel
  - OS saves process state (to PCB)
  - OS chooses new process to run
  - OS loads its state (from PCB)
  - Process is running
- Main source of overhead for a scheduling policy

Simplifying Assumptions

- One process per user
- One thread per process
  - More on this topic next week
- Processes are independent

Researchers developed these algorithms in the 70s when these assumptions were more realistic, and it is still an open problem how to relax these assumptions.

iClicker Question

Will a fair scheduling policy maximize throughput?

A. Yes
B. No

Scheduling Algorithms

- **FCFS**: First Come, First Served
- **Round Robin**: Use a time slice and preemption to alternate jobs.
- **SJF**: Shortest Job First
- **Multilevel Feedback Queues**: Round robin on priority queue.

First-Come-First-Served (FCFS)

- Also known as First-In-First-Out (FIFO)
- Scheduler executes jobs in arrival order
- Jobs run until they either complete or block on I/O
- In early FCFS schedulers, the job did not relinquish the CPU even when it was doing I/O.

Round Robin

- Add a timer and use a pre-emptive policy
- Run each process for its time slice (*scheduling quantum*)
- After each time slice, move the running process to the back of the queue
- Selecting a time slice:
  - Too large: waiting time suffers, degenerates to FCFS if processes are never preempted
  - Too small: throughput suffers because too much time is spent context switching
  - Balance the two by selecting a time slice where context switching is roughly 1% of the time slice.

Most time sharing systems use some variation of this policy.

A typical time slice today is between 10-100 milliseconds, with a context switch time of 0.1 to 1 millisecond.

iClicker Question

Is Round Robin more fair than FCFS?

A. Yes
B. No

SJF / SRTF: Shortest Job First

Shortest Job First (SJF):

- Schedule the job that has the least amount of work (measured in CPU time) to do until its next I/O request or termination.
  - I/O bound jobs get priority over CPU bound jobs.
- Works for preemptive and non-preemptive schedulers.

Shortest Remaining Time First (SRTF):

- Preemptive SJF

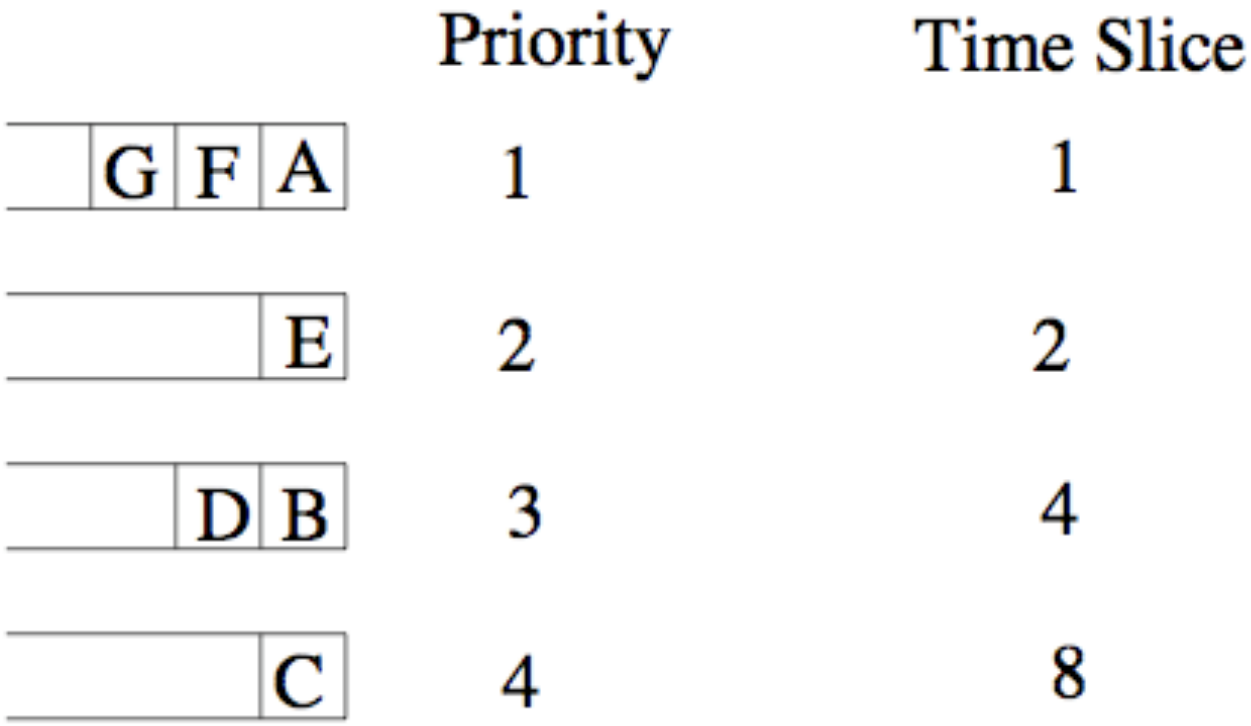
Using the Past to Predict the Future

- If a process is I/O bound in the past, it is also likely to be I/O bound in the future (programs turn out not to be random).
- To exploit this behavior, the scheduler can favor jobs (schedule them sooner) when they use very little CPU time (absolutely or relatively), thus approximating SJF.
- This policy is **adaptive** because it relies on past behavior and changes in behavior result in changes to scheduling decisions.

Approximating SJF: Multilevel Feedback Queues

- Multiple queues with different priorities.
- OS uses Round Robin scheduling at each priority level, running the jobs in the highest priority queue first.
- Once those finish, OS runs jobs out of the next highest priority queue, etc.
  - Can lead to starvation if highest priority jobs run forever
- Round robin time slice increases exponentially at lower priorities.

Multilevel Feedback Queues: Example



Approximating SJF: Multilevel Feedback Queues

Adjust priorities as follows (details can vary):

1. Job starts in the highest priority queue
2. If job's time slices expire, drop its priority one level.
3. If job's time slices do not expire (the context switch comes from an I/O request instead), then increase its priority one level, up to the top priority level.

==> In practice, CPU bounds drop like a rock in priority and I/O bound jobs stay at high priority

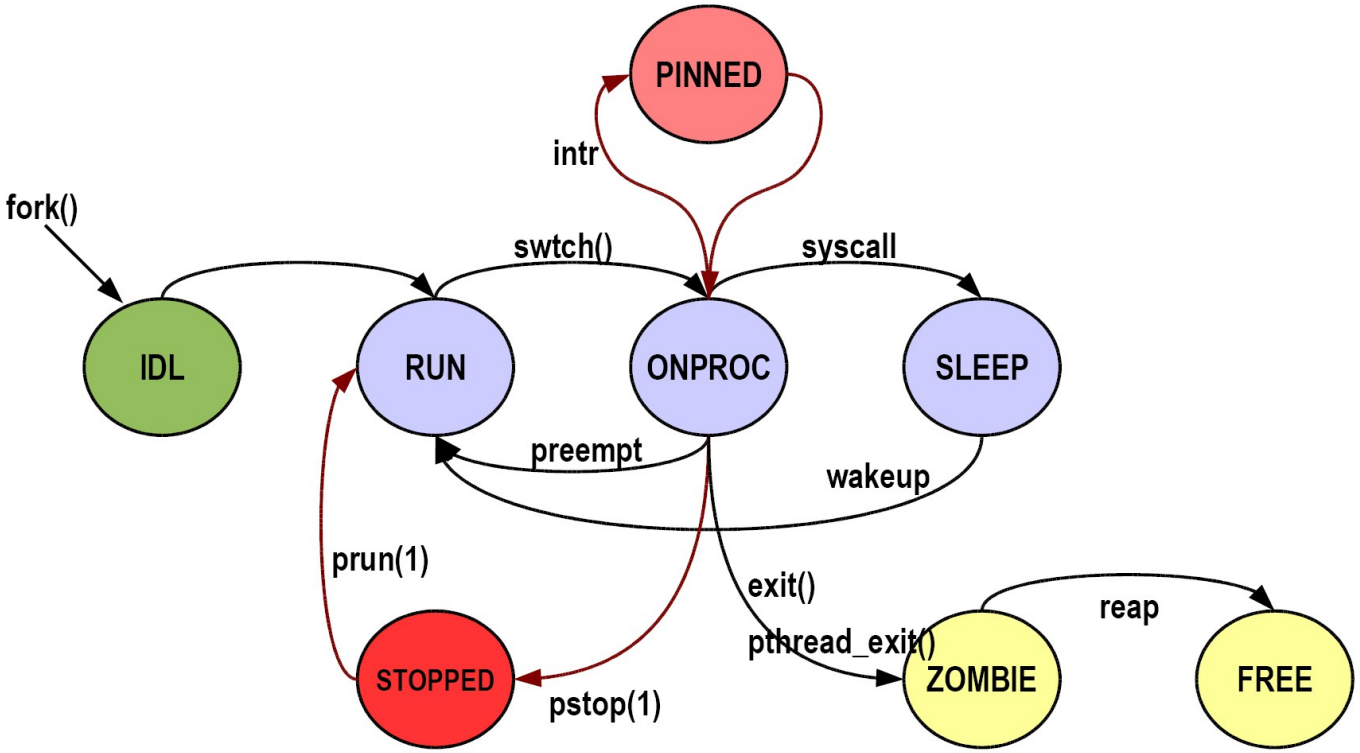
Improving Fairness

Since SJF is optimal, but unfair, any increase in fairness by giving long jobs a fraction of the CPU when shorter jobs are available will degrade average waiting time. Possible solutions:

- Give each queue a fraction of the CPU time. This solution is only fair if there is an even distribution of jobs among queues.
- Adjust the priority of jobs as they do not get serviced (Unix originally did this.) This ad hoc solution avoids starvation but average waiting time suffers when the system is overloaded because all the jobs end up with a high priority.

A Dose of Reality: Scheduling in Solaris

State Transitions



Close to our scheduling diagram, but more complicated

Something Different

Summary

The operating system scheduler schedules processes on the CPU

- Many algorithms are possible, most modern OSes use round robin

The scheduler can stop *any* process *any*where!

Announcements

- Next time: Threads and Too Much Milk
- Homework 2 due Friday at 8:45a
- Project 0 is up, due Friday, 2/13
  - Follow the instructions on Piazza to register your group
  - I'll begin harassing people not in groups tomorrow