

# CSC369: Operating Systems

CPU Scheduling

Andrew Petersen

# Anonymous Feedback

- *“A friend of mine does not like how you post on the discussion boards. He feels you come off a little mean and even cocky (I don't share his views on this topic but he seems really hurt). ...”*
- Ouch! Thanks for letting me know!
- That isn't my intent at all -- I do apologize.
  - I'm intending to be succinct and factual.
- It would be useful to have some examples.

# You're working me over!

- Yesterday, I had to ask Sue to help me recover from a nohup'd thread-bomb
- One of the tests we ran on AI caused one simulation to spawn -lots- of threads.
- Also yesterday: “*I wonder if the utm submit is vulnerable to curious inputs*”

*It totally is.”*

- Less so, now.  
 $\$value \sim s/^{([A-Za-z!?,.])}/+/eg$

# Administrivia

- We're behind on AI
  - Assignments have been automarked
  - Ioan is finishing up reviewing code in the last assignments
  - Marks out soon -- sorry for the delay!
- We're waiting on StG automarking for the speed race

# Office Hours Oct 12-16

- Monday is Thanksgiving!
- Wednesday, office hours are different from normal: 2:00-3:00 p.m.

# Exercise 2

- How was exercise 2?
- Any questions about system calls?
- System calls and OS/161?

# Assignment 2

- A2 will be released tonight.
- First: Complete the PID monitor by adding functions to the module.
- Second: Write 4 process system calls  
get\_pid, wait\_pid, fork, execv
- *Warning:* OS/161 uses the term “thread” when it means “process”

# And on to a new topic ...

- So far, we have:
  - Discussed how to exploit parallelism using processes and threads
  - Discovered the need for synchronization
    - ... and developed strategies for providing it, including locks, semaphores, and monitors
- Today, we'll talk about how to allocate the processor resource: cpu scheduling
- We'll also discuss deadlock management

# Outline

- What is CPU Scheduling?
- Types of Schedulers
- Basic Scheduling Heuristics
- Advanced Heuristics
- Real-world Schedulers
- Deadlock

# What is Processor Scheduling?

- “The allocation of processors to processes over time”
  - “Who gets to execute when?”
- This is the key to multiprogramming
  - We want to increase CPU utilization and job throughput
- Mechanisms: process states, process queues
- Policies:
  - Given more than one runnable process, how do we choose which to run next?
  - When do we make this decision?

# Scheduling Goals

- All systems
  - Fairness - each process receives fair share of CPU
    - Avoid starvation
  - Policy enforcement - usage policies should be met
  - Balance - all parts of the system should be busy
- Batch systems
  - Throughput - maximize jobs completed per hour
  - Turnaround time - minimize time between submission and completion
  - CPU utilization - keep the CPU busy all the time

# Goals, continued

- Interactive Systems
  - Response time - minimize time between receiving request and starting to produce output
  - Proportionality - “simple” tasks complete quickly
- Real-time systems
  - Meet deadlines
  - Predictability
- Goals often conflict with each other!
  - We need different schedulers for different systems

# The Life Cycle of a Process

- Processes repeatedly alternate between computation and I/O (reading files, accessing memory)
  - Called CPU bursts and I/O bursts
  - Last CPU burst ends with a call to terminate the process (`_exit()` or equivalent)
- During I/O bursts, the CPU is not needed
  - We have an opportunity to execute another process!

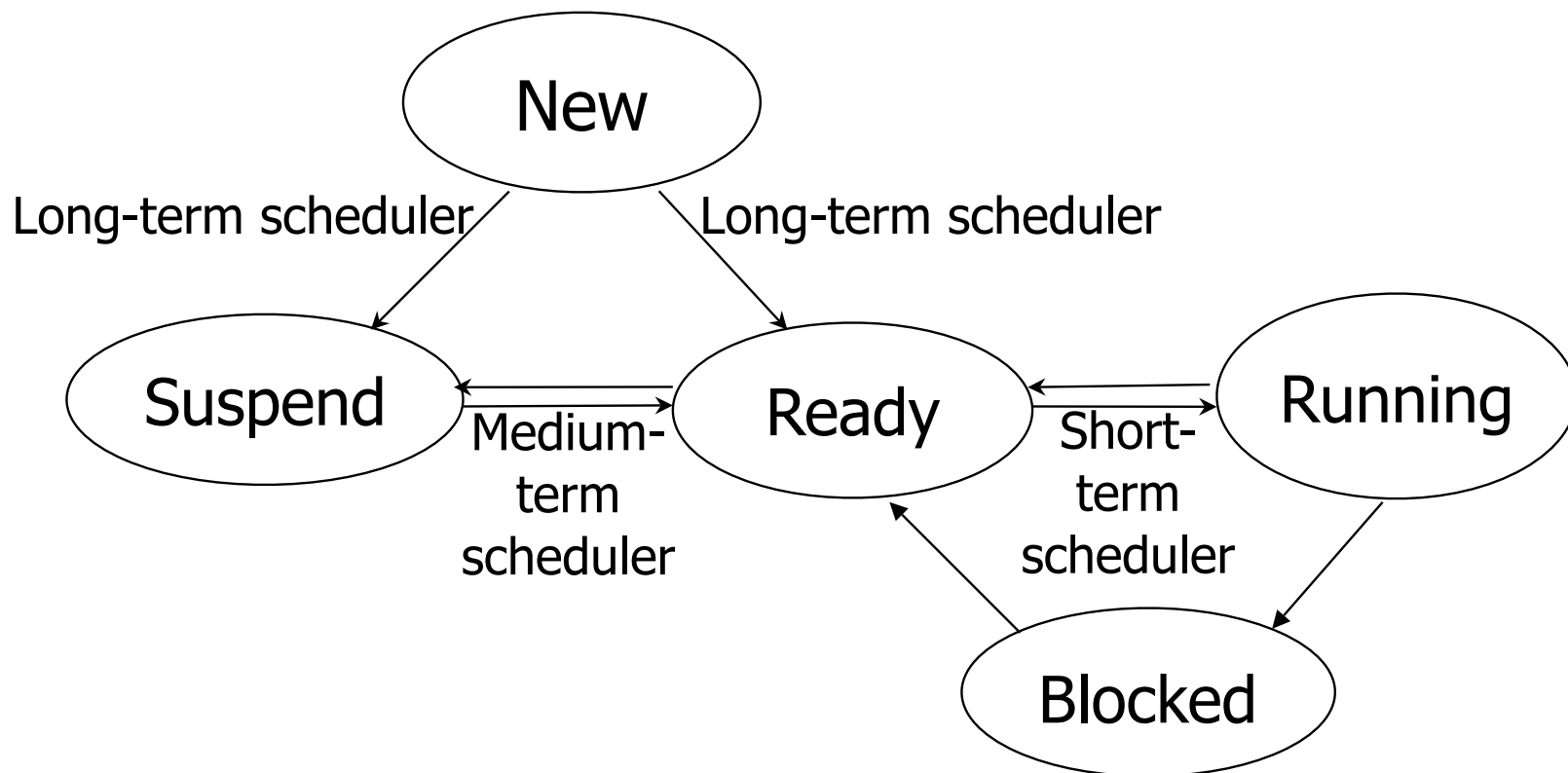
# Aside: Limits on Performance

- At some point, you will need to optimize an application
  - This leads to profiling
  - System architects break programs into two major classes:
    - CPU-bound: very long CPU bursts, infrequent I/O bursts
    - I/O-bound: short CPU bursts, frequent (long) I/O bursts
- What can you do about CPU-bound applications?
- What about I/O bound applications?

# Outline

- What is CPU Scheduling?
- Types of Schedulers
  - Short, medium, and long-term
  - Preemptive and nonpreemptive
- Basic Scheduling Heuristics
- Advanced Heuristics
- Real-world Schedulers
- Introduction to Memory Management

# Process State Diagram



# Types of CPU Scheduling

- Long-term scheduling (admission scheduling/control)
  - Used in batch systems, not common today
- Medium-term scheduling (memory scheduling)
  - A common but infrequent task
  - Decides which processes are swapped out to disk
  - We'll talk about this later in memory management
  - Sometimes called “long-term”, and admission control is ignored
- Short-term scheduling (dispatching)
  - Occurs frequently
  - Needs to be efficient (fast context switches, fast queue manipulation)

# Review: What Happens on Dispatch (Context Switch)?



# Review: What Happens on Dispatch (Context Switch)?

- Save currently running process state
  - Unless the current process is exiting
- Select next process from ready queue
  - Insert your favorite scheduling heuristic here!
- Restore state of next process
  - Restore registers
  - Restore OS control structures
  - Switch to user mode
  - Set PC to next instruction in the process

# When to Dispatch

- When a process enters the ready state
  - I/O interrupts
  - Signals
  - Process creation (or admission)
- When the running process blocks (or exits)
  - Operating system calls (e.g., I/O)
  - Signals
- At fixed intervals
  - Clock interrupts
    - See kern/thread/hardclock.c

# Types of Scheduling

- Non-preemptive scheduling
  - Once the CPU has been allocated to a process, it keeps the CPU until it terminates or blocks
  - Suitable for batch scheduling
- Preemptive scheduling
  - CPU can be taken from a running process and allocated to another
  - Needed in interactive or real-time systems

# Outline

- What is CPU Scheduling?
- Types of Schedulers
- Basic Scheduling Heuristics
  - FCFS, SJF, Round-Robin, Priority
- Advanced Heuristics
- Real-world Schedulers
- Deadlock

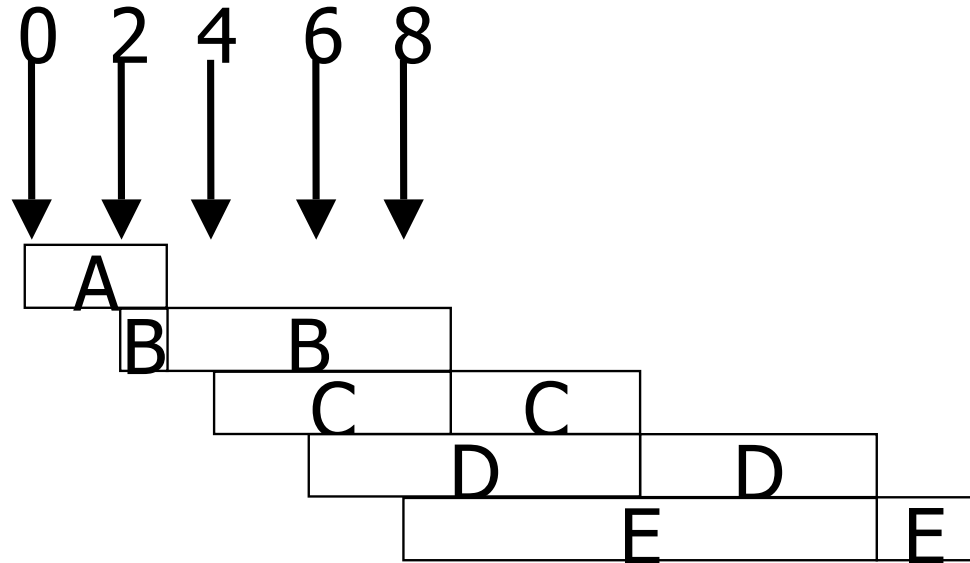
# Scheduling Algorithms: FCFS



- “First come, first served”
- Non-preemptive
- Choose the process at the head of the FIFO queue of ready processes
- Average waiting time under FCFS is often quite long
- convoy effect: all other processes wait for the one big process to release the CPU

# FCFS Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



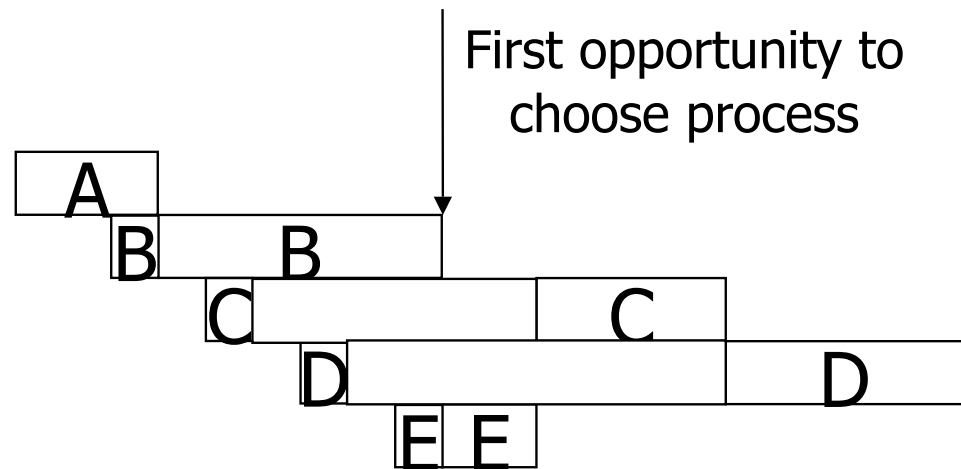
- Note that E waits five times as long as it runs!
- Total run time is 20
- Total wait time is 23, average wait is 4.6

# Algorithm: Shortest-Job-First

- aka Shortest Process Next, SJF or SPN
- Choose the process with the shortest expected processing time ... judged somehow:
  - Programmer estimate
  - History statistics
  - Can be shortest-next-CPU-burst for interactive jobs
- Provably optimal for “average wait time”

# Example: SJF

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



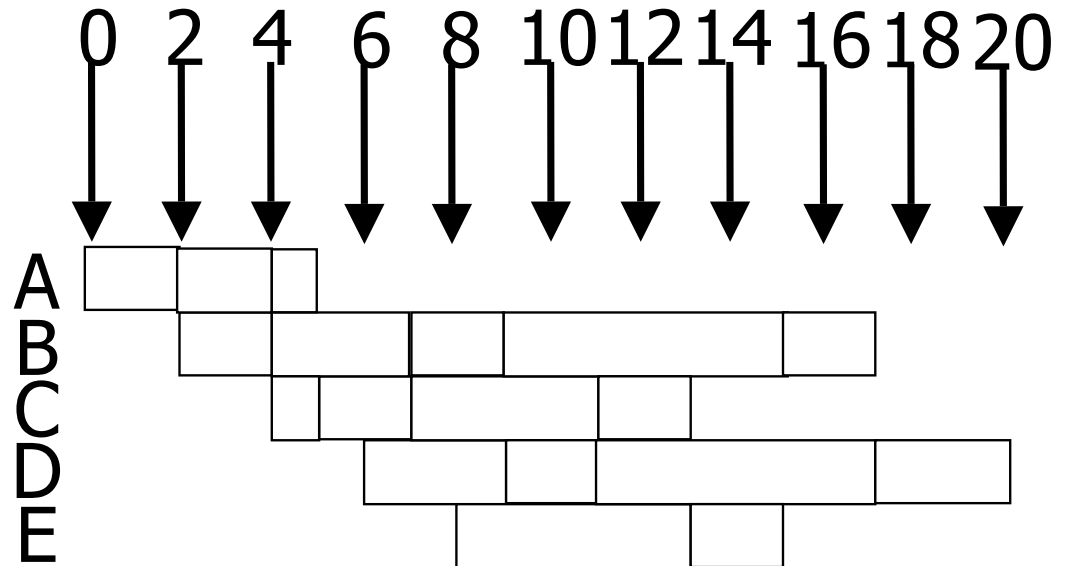
- Total run time still 20
- Total wait time is now 18?, average wait time now 3.4

# Algorithm: Round Robin

- Designed for time-sharing systems
- Preemptive
- Ready queue is circular
  - Each process runs for time quantum  $q$  before being preempted and placed on the queue
- Choice of quantum (aka time slice) is critical
  - as  $q \rightarrow \infty$ , RR  $\rightarrow$  FCFS; as  $q \rightarrow 0$ , RR  $\rightarrow$  processor sharing (PS)
  - we want  $q$  large w.r.t. the context switch time

# Example: Round-Robin

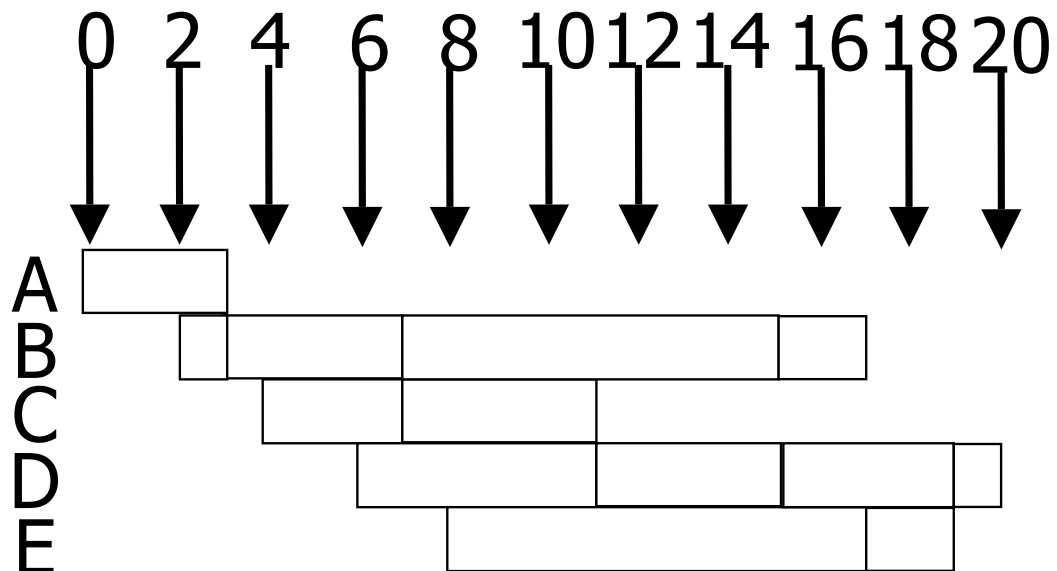
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



- Using a quantum of 2
- Assuming new processes arrive before running process is evicted

# Example: Round-Robin

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



- Very different with a quantum of 4!

# Algorithm: Priority Scheduling

- A priority  $p$  is associated with each process
- The highest priority job is selected from the Ready queue
  - Can be preemptive or non-preemptive
- Enforcing this policy is tricky
  - A low priority task may prevent a high priority task from making progress by holding a resource (priority inversion)
  - A low priority task may never get to run (starvation)

# Outline

- What is CPU Scheduling?
- Types of Schedulers
- Basic Scheduling Heuristics
- Advanced Heuristics
  - Queue, Feedback, and Fair-Share
- Real-world Schedulers
- Deadlock

# Algorithm: Multi-Level Queue Scheduling

- Have multiple ready queues
  - Each runnable process is on only one queue
- Processes are permanently assigned to a queue
  - Criteria include job class, priority, etc.
- Each queue has its own scheduling algorithm
  - Another level of scheduling decides which queue to choose next
  - Usually priority-based

# Algorithm: Feedback Scheduling

- Adjust the criteria for choosing a particular process based on past history (dynamic algorithm!)
- Can boost priority of processes that have waited a long time (aging)
- Can prefer processes that do not use full quantum
- Can boost priority following a user-input event
- Can adjust expected next-CPU-burst
- Combine with queue scheduling to move processes between queues

# Algorithm: Fair Share Scheduling

- Group processes by user or department
- Ensure that each group receives a proportional share of the CPU
  - Shares do not have to be equal
- Priority of a process depends on own priority and past history of entire group
- Variant: Lottery scheduling - each group is assigned “tickets” according to its share
  - Hold a lottery to find next process to run

# Exercise

- 4 processes (P0-P3) are being run
  - Each process  $P_i$  starts at time  $2 * i$
  - Each process does has a 3-unit CPU burst, a 2-unit I/O burst, and then a 5-unit CPU burst
- The scheduler is a 3-queue (Q0-Q2) priority scheduler (Q0 is the highest priority)
  - New processes and processes returning from I/O start in Q0
- Each queue uses round-robin with a quantum of 2
- If a process is preempted, it moves from queue  $i$  to queue  $i + 1$

# Coming Up ...

- Today, we ...
  - Reviewed how system calls are linked to programs and executed
  - Discussed various schemes for fairly allocating the processor resource
- Next week, we'll ...
  - Finish up scheduling by looking at linux implementations
  - Discuss the problem of deadlock which arises when resources must be shared
  - Introduce the idea of memory allocation