CSC369: Operating Systems

Fall 2014

Andrew Petersen

... AI must be a tribulation

In moments like these I would hope that many of us do not stray from our daily prayers. Lest we forget..

The Thread's Prayer:

Our Andrew, who art in parallel,
hallowed be thy name,
Thy process come,
Thy will be done
on earth as it is in memory.

Give us this day our daily compiles
and forgive us our seg faults
as we forgive those who seg fault against us,
and lead us not into plagiarism
but deliver us from malloc

Amen.

Haskell Workshop

- Abbas is running a Haskell workshop on either Oct 15 or 20

- Register here:

  http://goo.gl/5ONJNA

Outline

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

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

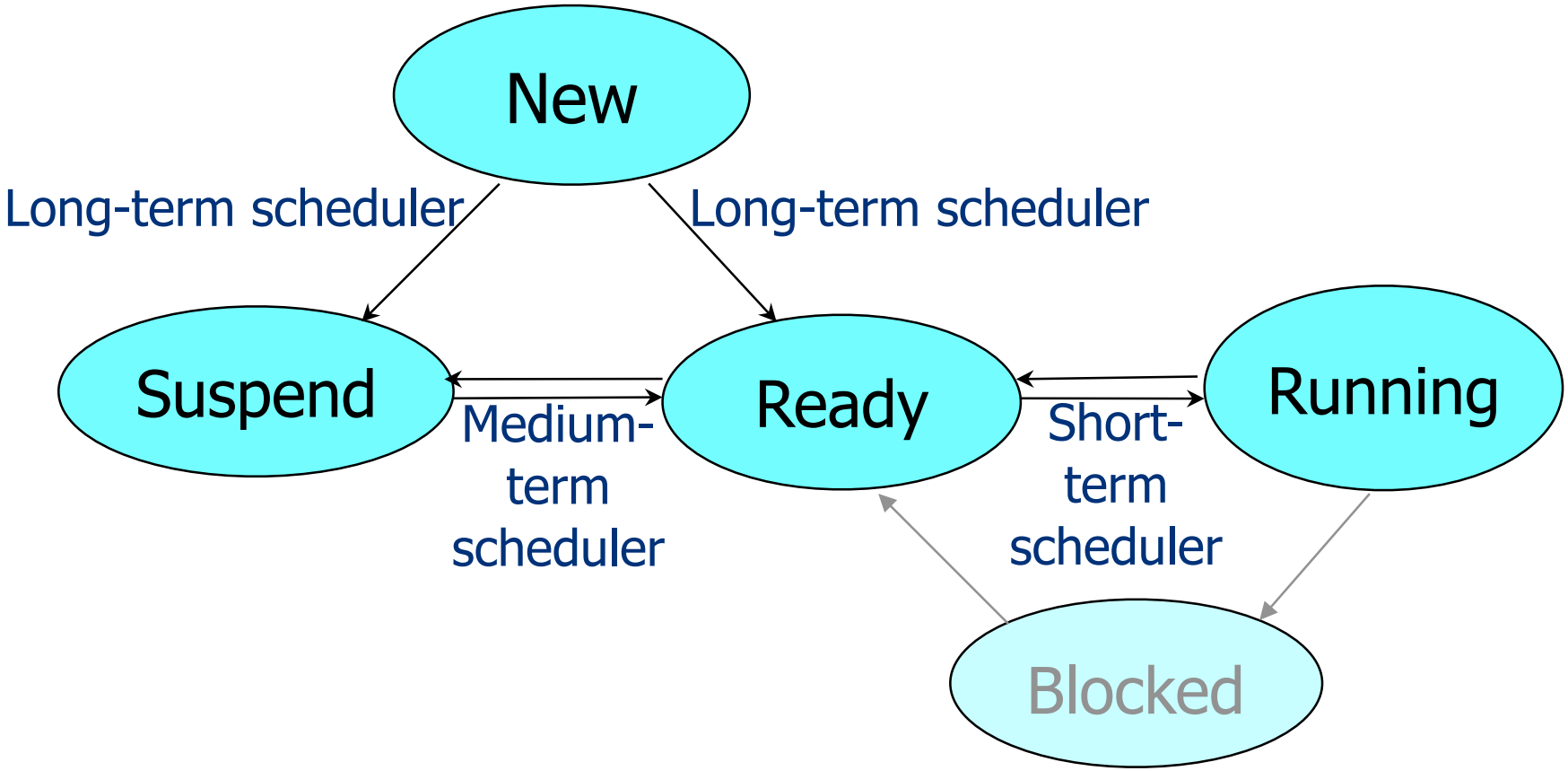
- At some point, you will need to optimize an application
  - This leads to profiling
- System architects break programs into two major classes:
  - CPU-bound: very long CPU bursts, infrequent I/O bursts
  - I/O-bound: short CPU bursts, frequent (long) I/O bursts

- 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

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

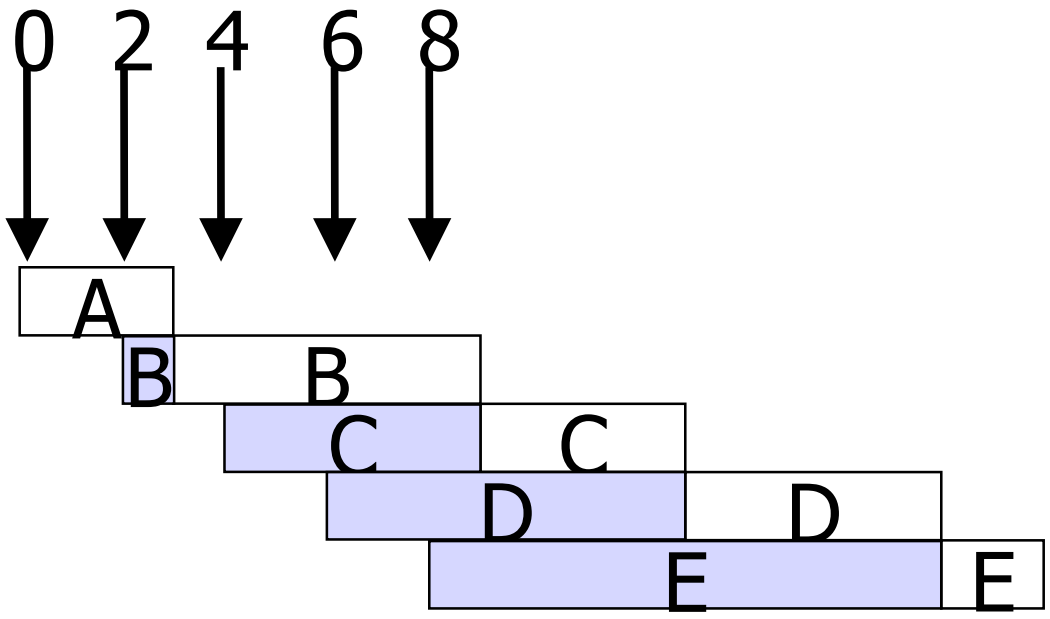
- What is CPU Scheduling?
- Types of Schedulers
- Basic Scheduling Heuristics
  - FCFS, SJF, Round-Robin, Priority
- Advanced Heuristics
- Real-world Schedulers

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



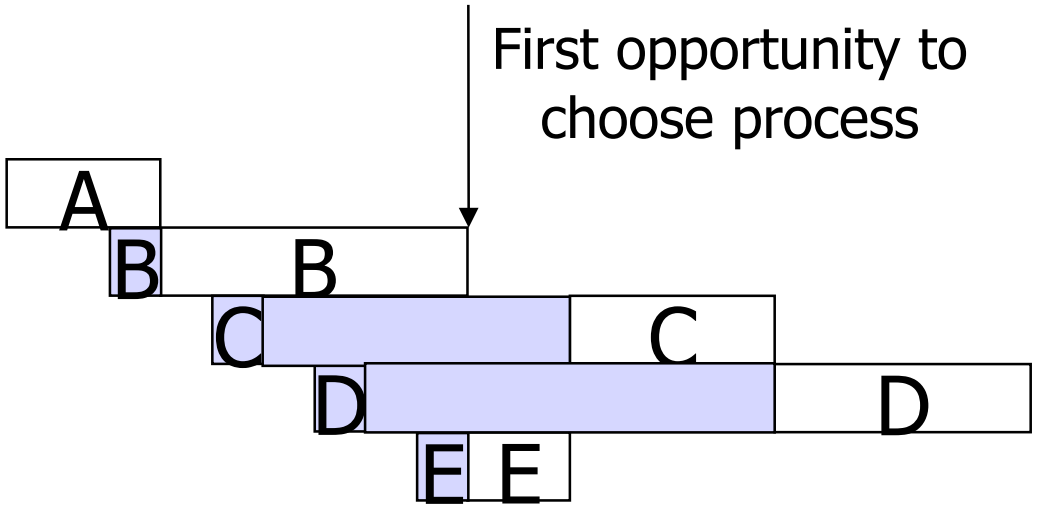
- Note that E waits five times as long as it runs!
  - Total run time is 20
  - Total wait time is 23, average wait is 4.6

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



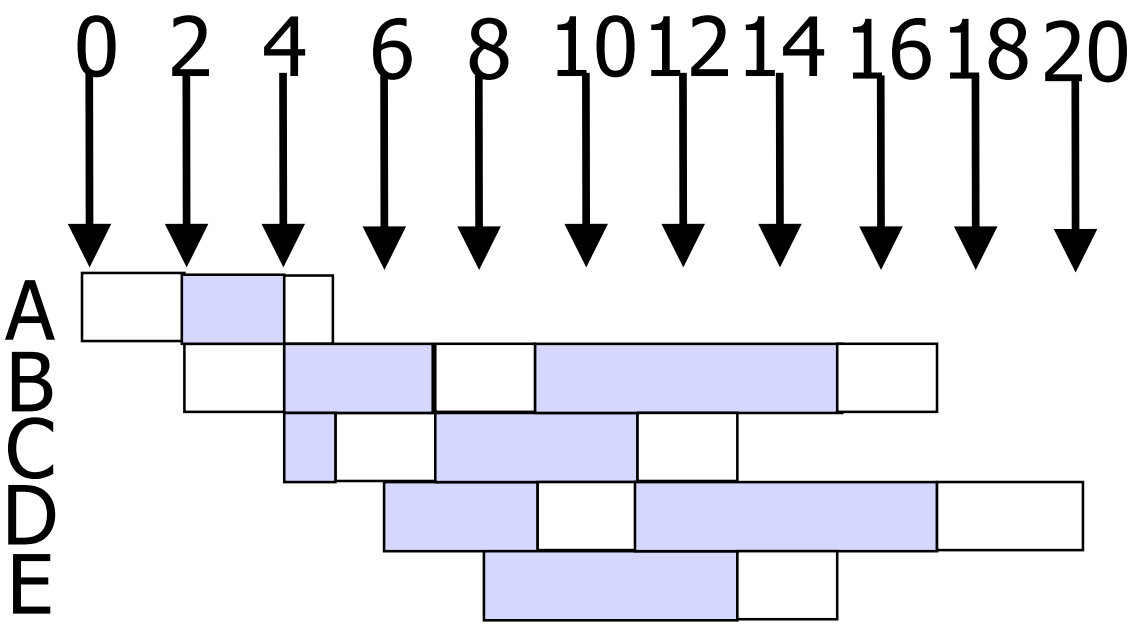
- Total run time still 20
- Total wait time is now 18?, average wait time now 3.4

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

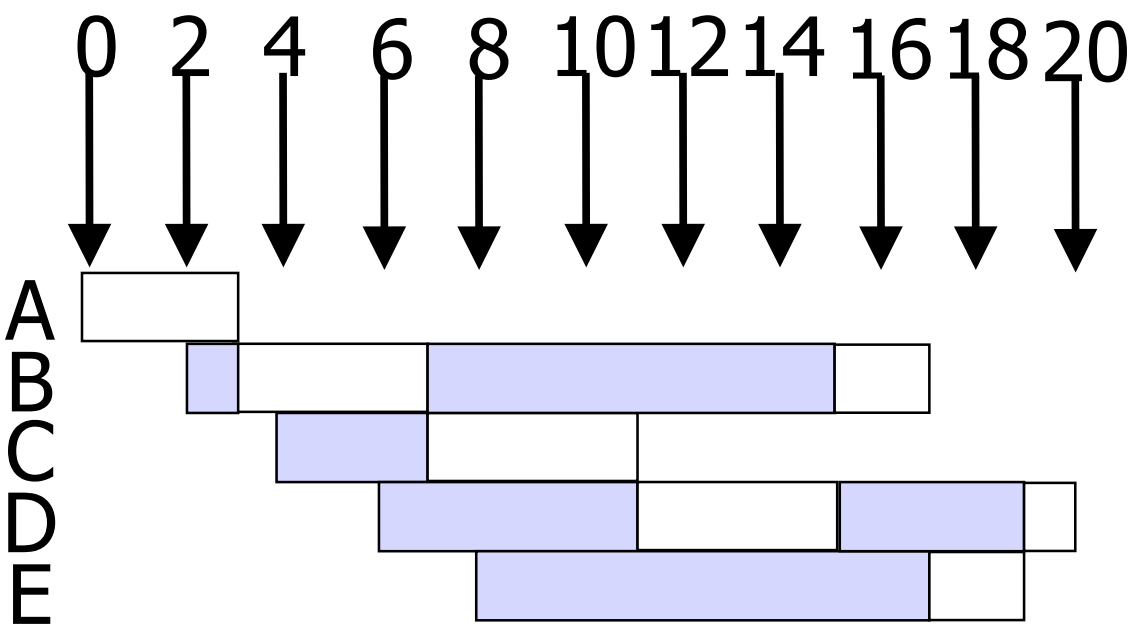
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



- 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

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 Pi 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

Outline

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

Unix CPU Scheduling

- Interactive processes are favored
  - Small CPU time slices are given to processes by a priority algorithm that reduces to Round Robin for CPU-bound jobs
- The more CPU time a process accumulates, the lower its priority becomes (negative feedback)
- "Process aging" prevents starvation
- Newer Unixes reschedule processes every 0.1 seconds and recompute priorities every second

UNIX Scheduling Details

- Multi-level Feedback Queue with Round Robin within each priority queue
- Priority is based on process type and execution history
  $P_j(i) = base_j + [CPU_j(i-1)]/2 + nice_j$

  $CPU_j(i) = U_j(i)/2 + [CPU_j(i-1)]/2$
  - $P_j(i)$: priority of process j at beginning of interval i; lower values equal higher priorities
  - $base_j$: base priority of process j
  - $U_j(i)$: processor utilization of process j in interval i
  - $CPU_j(i)$: exponentially weighted average processor utilization by process j through interval i
  - $nice_j$: user-controllable adjustment factor

(Old) Linux CPU Scheduling

- 2 separate process scheduling algorithms:
  - time-sharing and real-time tasks
- Time-sharing: use a prioritized, credit-based algorithm
  - Chooses process with the most credits
  - At every timer interrupt, the running process loses one credit
  - When credits reach 0, the process is suspended
  - If no runnable processes have any credits, linux performs a recrediting, adding credits to *every* process: credits = credits/2 + priority

Linux Scheduling Details

- Re-credit step takes time proportional to the number of processes – O(N)
  - For large scale systems, spend too much time making scheduling decisions
- 2.5 kernel introduced O(1) scheduler
  - Each process gets a time quantum, based on its priority
  - Two arrays of runnable processes, active and expired
    - Processes are selected to run from the active array
    - When quantum exhausted, process goes on expired
    - When active is empty, swap the two arrays

Windows NT CPU Scheduling

- Dispatcher uses a 32-level MLFQ priority scheme
  - The real-time class has threads with priorities 16 - 31; the variable class has threads with priorities 0 - 15
- Dispatcher traverses the queues from highest to lowest until it finds a ready thread
- When a variable class thread's quantum expires, its priority is lowered; when it is released from a wait, its priority is raised

Windows Scheduling Details

- Preemptive scheduler
- Real-time:
  - All threads have a fixed priority
    - At a given priority, processes are in a RR queue
- Variable:
  - A thread's priority begins at some initial assigned value but may change
    - There is a FIFO queue at each priority level
  - If it has used up its current time quantum, NT lowers its priority
  - If it is waiting on an I/O event, NT raises its priority (more for interactive waits than for other types of I/O waits)

Scheduling Summary

- Scheduling is one example of sharing a fixed resource
  - Influences process/thread performance and impacts synchronization
- Several criteria must be considered: fairness, performance, real-time demands, ...
- Most "real" implementations are hybrids of a handful of theoretical algorithms