

---

## Process Scheduling

CSE 410, Spring 2007  
Computer Systems

<http://www.cs.washington.edu/410>

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

1

---

## Readings and References

- Reading
  - » *Operating System Concepts*
    - Chapter 6, Sections 6.1 through 6.5
    - Chapter 6, Section 6.7.2, Windows scheduling example
- Other References
  - » Chapter 6, Section “Thread Scheduling”, *Inside Microsoft Windows 2000*, Third Edition, Solomon and Russinovich

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

2

---

## Process State

- A process can be in one of several states
  - » new, ready, running, waiting, terminated
- The OS keeps track of process state by maintaining a queue of PCBs for each state
- The ready queue contains PCBs of processes that are waiting to be assigned to the CPU

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

3

---

## Windows 2000 Thread States

- 7 - Unknown
- 6 - Transition
- 5 - Wait (for something to complete)
- 4 - Terminated
- 3 - Standby (on-deck circle)
- 2 - Running (at bat)
- 1 - Ready (eligible to be selected)
- 0 - Initialized

ThreadStates.X1.msc

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

4

## The Scheduling Problem

- Need to share the CPU between multiple processes in the ready queue
  - » OS decides which process gets the CPU next
  - » Once a process is selected, OS does some work to get the process running on the CPU

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

5

## How Scheduling Works

- The short-term scheduler is responsible for choosing a process from the ready queue
- The scheduling algorithm implemented by this module determines how process selection is done
- The scheduler hands the selected process off to the dispatcher which gives the process control of the CPU

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

6

## Scheduling Decisions - When?

- Scheduling decisions are always made:
  - » when a task is terminated
  - » when a task switches from running to waiting
- Scheduling decisions are also made when an interrupt occurs in a preemptive system

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

7

## Scheduling Decisions - Why?

- Maximize throughput and resource utilization
  - » Need to overlap CPU and I/O activities.
- Minimize response time, waiting time and turnaround time
- Share CPU in a “fair” way
- Conflicting constraints
  - » constantly need to make tradeoffs

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

8

## Scheduling Decisions – How?

- Non-preemptive scheduling
  - » The task decides when it stops
  - » The scheduler must wait for a running task to voluntarily relinquish the CPU
  - » Used in the past, now only in real-time systems
- Preemptive scheduling
  - » OS can force a running task to give up control of the CPU and pick another task to run
  - » Used by all major OS's today

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

9

## Non-preemptive scheduling

- Non-preemptive scheduling
  - » The scheduler waits for a running task to voluntarily relinquish the CPU (task either terminates or blocks)
- Simplifies kernel
- Simplifies hardware
- But it also makes it difficult to manage the system's performance effectively with arbitrary processes running

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

10

## Preemptive scheduling

- Preemptive scheduling
  - » The OS can force a running task to give up control of the CPU, allowing the scheduler to pick another task
  - » OS gains control on a regular interrupt schedule
- A little more overhead
- But allows much better control of the overall system performance when running an arbitrary selection of processes

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

11

## CPU and I/O Bursts

- Typical process execution pattern:
  - » use the CPU for a while (CPU burst)
  - » then do some I/O operations (I/O burst)
- *CPU bound* processes have long CPU bursts and perform I/O operations infrequently
- *I/O bound* processes spend most of their time doing I/O and have short CPU bursts

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

12

## First Come First Served

- Scheduler selects the process at the head of the ready queue; typically non-preemptive
- Example: 3 processes arrive at the ready queue in the following order:

P1 ( CPU burst = 240 ms), P2 ( CPU burst = 30 ms),

P3 ( CPU burst = 30 ms)



- + Simple to implement
- Average waiting time can be large

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

13

## Round Robin

- FCFS + preemptive scheduling
- Ready queue is a circular queue
- Each process gets the CPU for a time quantum (a time slice), typically 10 - 100 ms
- A task runs until it uses up its time slice or blocks

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

14

## Round Robin Examples

- Short jobs don't get stuck behind long jobs



- Average response time for jobs of same length is bad



5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

15

## Round Robin Pros and Cons

- + Works well for short jobs; typically used in timesharing systems
- Higher overhead due to frequent context switches
- Increases average waiting time, especially if CPU bursts are the same length and need more than one time quantum

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

16

## Priority Scheduling

- Select the process with the highest priority
- Priority is based on some attribute of the process (e.g., memory requirements, owner of process, etc.)
- Starvation problem
  - » low priority jobs may wait indefinitely
  - » can prevent starvation by **aging** (increase process priority as it waits)

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

17

## Priority Inversion

- Three tasks with priorities:  $A_{HI}$ ,  $B_{MED}$ ,  $C_{LOW}$
- Suppose  $C_{LOW}$  locks resource that  $A_{HI}$  needs
  - »  $C_{LOW}$  prevents  $A_{HI}$  from running
  - »  $B_{MED}$  prevents  $C_{LOW}$  from running
  - »  $A_{HI}$  can't run until  $B_{MED}$  finishes and  $C_{LOW}$  unlocks
- This is known as **priority inversion**
- Solution: increase priority of a process holding a lock to the max priority of a process waiting on the lock
  - »  $C_{LOW} \rightarrow C_{HI}$  until it releases the lock

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

18

## Shortest Job First

- Special case of priority scheduling
  - » priority = expected length of CPU burst
- Scheduler chooses the process with the shortest remaining time to completion
  - » think about waiting at the copy machine
- Example: What's the average waiting time?



5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

19

## Shortest Job First Pros and Cons

- + It's the best you can do to minimize average response time
  - » can prove the algorithm is optimal
- Difficult to predict the future
  - » Use past behavior of the task to predict length of its next CPU burst
- Unfair-- possible starvation
  - » many short jobs can stall long jobs

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

20

## Multi-level Queues

---

- Maintain multiple ready queues based on task “type” (e.g., system, interactive, batch)
- Each task is assigned to a particular queue
  - » Each queue has a priority
  - » May use a different scheduling algorithm in each queue
  - » There are policies implicit in these choices
- Also need to schedule between queues

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

21

## Multi-level Feedback Queues

---

- Adaptive algorithm: task priority changes based on past behavior
- Task starts with high priority
  - » because it’s probably a short job
- Decrease priority of tasks that hog the CPU (CPU-bound jobs)
- Increase priority of tasks that don’t use the CPU much (I/O-bound jobs)

5/10/2007

cse410-21-scheduling © 2006-07 Perkins, DW Johnson and University of Washington

22