

## CS 414 : Operating Systems

UNIVERSITY OF VIRGINIA  
Department of Computer Science

Spring 2008

### Topic 8: CPU Scheduling

- Readings for this topic: Chapter 5.
- OS typically consists of two parts: 1) Process coordination, and 2) Resource Management
- Resources fall into two classes:
  - Preemptible vs Non-preemptible
  - Is this distinction absolute? Real issue?
- OS makes two related kinds of decisions about resources. What is the goal?
  - Allocation: who gets what. Implication?
  - Scheduling: how long can they keep it. Implication?
- Resource #1:
- Processes may be in any one of three general scheduling states:
  - Running.
  - Ready.
  - Blocked.
- Goals for scheduling disciplines:
  - Efficiency of resource utilization. (*keep CPU and disks busy*)
  - Minimize overhead. (*context switching*)
  - Minimize response time. *Define response time.*
  - Distribute cycles equitably. What does this mean?

- FIFO (also called FCFS): run until finished.
  - Usually, “finished” means “blocked”.
  - Problem?
- Solution: limit maximum amount of time that a process can run without a context switch. This time is called a *time slice*.
- Round Robin: run process for one time slice, then move to back of queue. Each process gets equal share of the CPU. What if the slice isn't chosen carefully?
  - *Too long:*
  - *Too small:*
- Originally, Unix had 1 sec. time slices. Right size?
 

Most systems today use time slices of 10,000 - 100,000 instructions.
- How to implement priorities in RR?
- Is RR always better than FIFO?
- What is the best we can do? Is there "perfect" scheduling algorithm? *STCF*: shortest time to completion first with preemption. In what sense is it the best?
- Example: two processes, one doing 1 ms computation followed by 10 ms I/O, one doing all computation. Suppose we use 100 ms time slice: I/O process only runs at 1/10th speed, I/O devices are only utilized 10% of time. Suppose we use 1 ms time slice: then compute-bound process gets interrupted 9 times unnecessarily for each valid interrupt. *STCF* works well.
- Why not using *STCF*?
- Rule of thumb: Give the most to those who need the least. What's the idea here?
- The strategy?

- Exponential Queue (or Multi-Level Feedback Queues):
  - Give newly runnable process a high priority and a very short time slice. If process uses up the time slice without blocking then decrease priority by 1 and double time slice for next time.
  - Go through the above example, where the initial values are 1ms and priority 100.
  - Techniques like this one are called *adaptive*. They are common in interactive systems.
  - The CTSS system (MIT, early 1960's) was the first to use exponential queues.
- Fair-share scheduling:
  - Keep history of recent CPU usage for each process.
  - Give highest priority to process that has used the least CPU time recently. Highly interactive jobs, like vi, will use little CPU time and get high priority. CPU-bound jobs, like compiles, will get lower priority.
  - Can adjust priorities by changing “billing factors” for processes. E.g. to make high-priority process, only use half its recent CPU usage in computing history.
- Performance evaluation of scheduling algorithms:
  - Analytic methods:
    - deterministic modeling
    - queueing model
  - simulation
  - implementation
- Summary:
  - In principle, scheduling algorithms can be arbitrary, since the system should produce the same results in any event.
  - Scheduling algorithms have strong effects on the the system's overhead, efficiency, and response time.
  - The best schemes are adaptive. To do absolutely best, we need to predict the future.