CS323 Operating Systems

CPU Scheduling III

Yuanyuan Zhou

Lecture 7

2/5/2003

Content of this lecture

- Administrative announcements
- Scheduling algorithms
  - Shortest process time
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling
- Scheduling for
  - Multiple processors
  - Threads
  - Real-time systems
- Summary

Administrative

- Quiz1due Friday 5pm
- MP1(thread scheduling)

Review: Scheduling Algorithms

- Batch systems
  - First come first serve
  - Shortest job first
- Interactive systems
  - Round-robin
  - Priority scheduling
  - Multi-Queue & Multi-level feedback

Shortest Process Time

- Based on shortest job first
- Changed for interactive systems
  - Consider the average response time for each user input/command
  - Estimated based on past response
    - Average time in previous runs: T_0, T_1, T_2, T_3
    - The average time for the next run is predicted to be $T_0 /8 + T_1 /8 + T_2 /4 + T_3 /2$
- What problem does it have?

Guaranteed Scheduling (QoS)

- Make real promises to the users about performance and then live up to them
- Example:
  - with n processes running, the scheduler makes sure that each one gets 1/n of the CPU cycles.
- Scheduling:
  - compute the ratio of actual CPU time consumed to CPU time entitled
  - Select the one with the lowest ratio
- Can it lead to starvation?

Lottery Scheduling

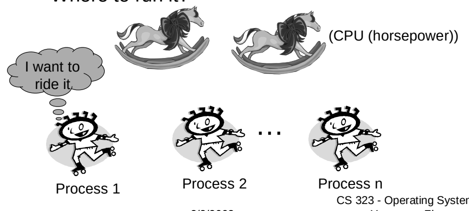
- More commonly used
- Probability-based:
  - Give processes lottery tickets. At scheduling time, a lottery ticket is chosen at random, and the process holding that ticket gets that resource.
- Give more tickets for higher priority processes
- Advantages:
  - Simple
  - Highly responsive
  - Can support cooperation between processes
  - Easy to support priority and proportion requirement

Fair-Share Scheduling

- Is Round-robin fair?
  - Yes, it is (from process point of view)
  - No, it may be not (from user point view)
- User-based fair share scheduling
  - Each user gets fair share
- Example:
  - Alice has 4 processes: A1, A2, A3, A4
  - Bob has 1 process: B1
  - Then A1, A2, A3, A4 are entitled only to 50% CPU, while B1 alone is entitled to 50%

Multi-Processor Scheduling: Load Sharing

- Decides
  - Which process to run?
  - How long does it run
  - Where to run it?



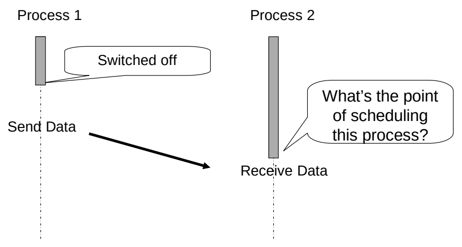
Multi-Processor Scheduling Choices

- Self-Scheduled
  - Each CPU dispatches a job from the ready queue
- Master-Slave
  - One CPU schedules the other CPUs
- Asymmetric
  - One CPU runs the kernel and the others runs the user applications.
  - One CPU handles network and the other handles applications

Gang Scheduling for Multi-Processors

- A collection of processes belonging to one job
- All the processes are running at the same time
  - If one process is preempted, all the processes of the gang are preempted.
- Helps to eliminate the time a process spends waiting for other processes in its parallel computation.

Why Gang Scheduling?



Scheduling in Real-Time Systems

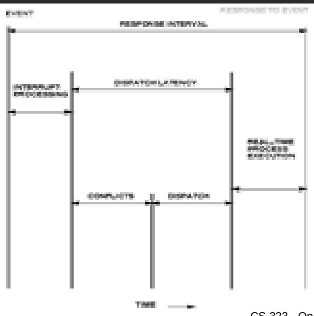
Schedulable real-time system. Given

- m periodic events
- event i occurs within period P_i and requires C_i seconds

- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \le 1$$

Latency in Dispatching



Dispatching Latency

- Goal: keep dispatch latency
- Problem: system call
  - small OS may enforce process to wait either for a system call to complete or for an I/O block to take place
- Solution: need preemptible system calls
  - insert preemption points (can be placed at safe location where kernel structures are not modified)
  - make the kernel preemptible (all kernel structures must be protected through the use of various synchronization mechanisms)

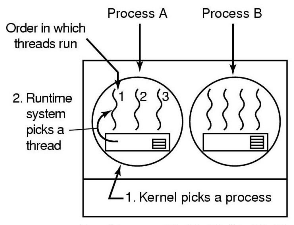
Priority Inversion and Inheritance

- Priority inversion
  - When a higher priority process needs to read or modify kernel data that are currently being accessed by a lower priority process.
  - The higher priority process must wait!
  - But the lower priority cannot proceed quickly due to scheduling.
- Solution: priority inheritance
  - When a lower-priority process accesses a resource, it inherits high-priority until it is done with the resource in question. And then its priority reverses to its natural value.

User-level Thread Scheduling

Possible Scheduling

- 50-msec process quantum
- run 5 msec/CPU burst

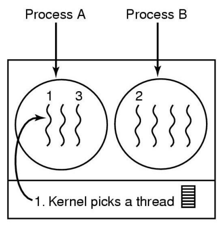

Process A
Process B
Order in which threads run
1
2
3
2. Runtime system picks a thread
1. Kernel picks a process

Possible:	A1, A2, A3, A1, A2, A3
Not possible:	A1, B1, A2, B2, A3, B3

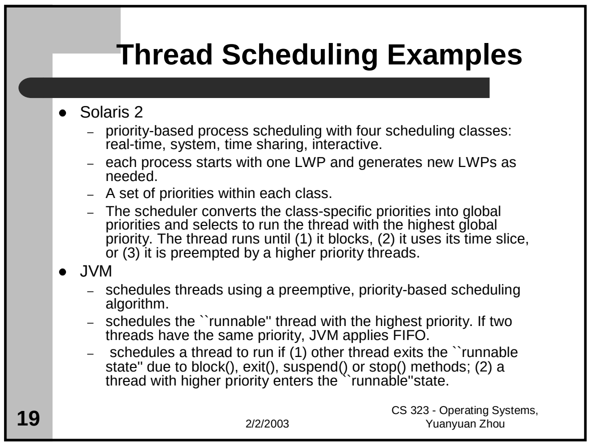
Kernel-level Thread Scheduling

Possible scheduling

- 50-msec process quantum
- threads run 5 msec/CPU burst



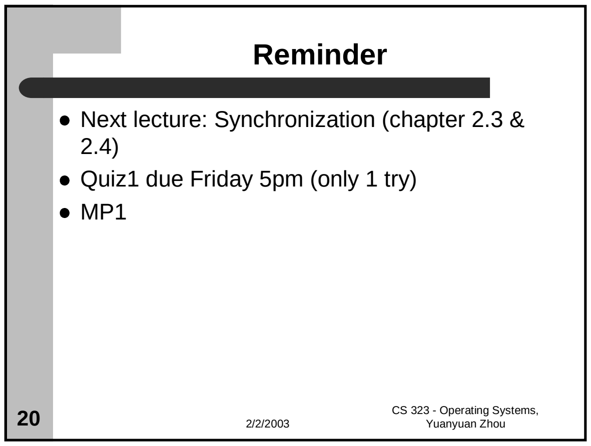
Possible:	A1, A2, A3, A1, A2, A3
Also possible:	A1, B1, A2, B2, A3, B3



Thread Scheduling Examples

- Solaris 2
  - priority-based process scheduling with four scheduling classes: real-time, system, time sharing, interactive.
  - each process starts with one LWP and generates new LWPs as needed.
  - A set of priorities within each class.
  - The scheduler converts the class-specific priorities into global priorities and selects to run the thread with the highest global priority. The thread runs until (1) it blocks, (2) it uses its time slice, or (3) it is preempted by a higher priority threads.
- JVM
  - schedules threads using a preemptive, priority-based scheduling algorithm.
  - schedules the ``runnable'' thread with the highest priority. If two threads have the same priority, JVM applies FIFO.
  - schedules a thread to run if (1) other thread exits the ``runnable state'' due to block(), exit(), suspend() or stop() methods; (2) a thread with higher priority enters the ``runnable''state.

19 2/2/2003 CS 323 - Operating Systems, Yuanyuan Zhou



Reminder

- Next lecture: Synchronization (chapter 2.3 & 2.4)
- Quiz1 due Friday 5pm (only 1 try)
- MP1

20 2/2/2003 CS 323 - Operating Systems, Yuanyuan Zhou