Today

• Finish comparing user and kernel threads
• Whether it be processes or threads that are the basic execution unit the OS needs to some way of scheduling each process time on the CPU
• Why?
  – To make better utilization of the system
Goals for Multiprogramming

• In a multiprogramming system, we try to increase utilization and throughput by overlapping I/O and CPU activities.
• This requires several os policy decisions:
  – Determine the multiprogramming level -- the number of jobs loaded in primary memory
  – Decide what job is to run next to guarantee good service
• These decisions are long-term and short-term scheduling decisions, respectively.
• Short-term scheduling executes more frequently, changes of multiprogramming level are more costly.

Scheduling

• The scheduler is a main component in the OS kernel
• It chooses processes/threads to run from the ready queue.
• The scheduling algorithm determines how jobs are scheduled.
• In general, the scheduler runs:
  – When a process/thread switches from running to waiting
  – When an interrupt occurs
  – When a process/thread is created or terminated
• In a preemptive system, the scheduler can interrupt a process/thread that is running.
• In a non-preemptive system, the scheduler waits for a running process/thread to explicitly block
Preemptive and Non-preemptive Systems

- In a non-preemptive system the OS will not stop a running jobs until the job either exists or does an explicit wait
- In a preemptive system the OS can potentially stop a job midstream in its execution in order to run another job
  - Quite often a timer going off and the current jobs time-slice or quantum being exhausted will cause preemption

Preemptive and Non-preemptive System (continued)

- I cannot over emphasize the need to understand the difference between preemptive and non-preemptive systems
- Preemptive systems also come in various degrees
  - Preemptive user but non-preemptive kernel
  - Preemptive user and kernel
- This affects your choice of scheduling algorithm, OS complexity, and system performance
Scheduling Algorithms Criteria

- There are many possible criteria for evaluating a scheduling algorithm:
  - CPU utilization
  - Throughput
  - Turnaround time
  - Waiting time
  - Response time
- In an interactive system, predictability may be more important than a low average but high variance.
- One OS goal is to give applications the illusion they are running unhindered by other jobs sharing the CPU and memory

Various Scheduling Algorithms

- First-come First-served
- Shortest Job First
- Priority scheduling
- Round Robin
- Multi-level queue

- We’ll examine each in turn
Scheduling Algorithms
First-Come First-Served

• **First-come First-served (FCFC) (FIFO)**
  – Jobs are scheduled in order of arrival to ready Q
  – Typically non-preemptive
• Problem:
  – Average waiting time can be large if small jobs wait behind long ones

```
Job A B C
B C   Job A
```

– May lead to poor overlap of I/O and CPU

Scheduling Algorithms
Shortest Job First

• **Shortest Job First (SJF)**
  – Choose the job with the smallest (expected) CPU burst
  – Provability optimal min. average waiting time
• Problem:
  – Impossible to know size of CPU burst (but can try to predict from previous activity)
• Can be either preemptive or non-preemptive
• Preemptive SJF is called *shortest remaining time first*
Scheduling Algorithms
Priority Scheduling

- Priority Scheduling
  - Choose next job based on priority
  - For SJF, priority = expected CPU burst
  - Can be either preemptive or non-preemptive

- Problem:
  - Starvation: jobs can wait indefinitely

- Solution to starvation
  - Age processes: increase priority as a function of waiting time

Scheduling Algorithms
Round Robin

- Round Robin
  - Used for timesharing in particular
  - Ready queue is treated as a circular queue (FIFO)
  - Each process is given a time slice called a \textit{quantum}
  - It is run for the quantum or until it blocks

- Problem:
  - Frequent context switch overhead
Scheduling Algorithms
Multi-level Queues

- Multi-level Queues
  - Probably the most common method used
  - Implement multiple ready Queues based on the job priority
  - Multiple queues allow us to rank each job’s scheduling priority relative to other jobs in the system
  - Windows NT/2000 has 32 priority levels
    - Each running job is given a time slice or quantum
    - After each time slice the next job of highest priority is given a chance to run
    - Jobs can migrate up and down the priority levels based on various activities

Next time

- With so much potentially going on in the system how do we synchronize all of it?