#### **CPU Scheduling Goals**

- CPU scheduler must decide:
  - How long a process executes
  - In which order processes will execute
- User-oriented scheduling policy goals:
  - Minimize average response time (time from request received until response starts) while maximizing number of interactive users receiving adequate response
  - Minimize turnaround time (time from process start until completion)
    - Execution time plus waiting time
  - Minimize variance of average response time
    - Predictability is important
    - Process should always run in (roughly) same amount of time regardless of the load on the system

Fall 2002, Lecture 18

#### **FCFS Evaluation**

- Non-preemptive
- Response time slow if there is a large variance in process execution times
  - If one long process is followed by many short processes, short processes have to wait a long time
  - If one CPU-bound process is followed many I/O-bound processes, there's a "convoy effect"
    - Low CPU and I/O device utilization
- Throughput not emphasized
- Fairness —penalizes short processes and I/O bound processes
- Starvation not possible
- Overhead minimal

#### **CPU Scheduling Goals (cont.)**

- System-oriented scheduling policy goals:
  - <u>Maximize</u> throughput (number of processes that complete in unit time)
  - <u>Maximize</u> processor utilization (percentage of time CPU is busy)
- Other (non-performance related) systemoriented scheduling policy goals:
  - Fairness in the absence of guidance from the user or the OS, processes should be treated the same, and no process should suffer starvation (being infinitely denied service)
    - May have to be less fair in order to minimize average response time!
  - Balance resources keep all resources of the system (CPU, memory, disk, I/O) busy
    - Favor processes that will underuse stressed resources

Fall 2002, Lecture 18

Fall 2002, Lecture 18

# Preemptive vs. Non-Preemptive Scheduling

- Non-preemptive scheduling scheduler executes only when:
  - Process is terminated
  - Process switches from running to blocked
- Preemptive scheduler scheduler can execute at (almost) any time:
  - Executes at times above, also when:
    - Process is created
    - Blocked process becomes ready
    - A timer interrupt occurs
  - More overhead, but keeps long processes from monopolizing CPU
  - Must not preempt OS kernel while it's servicing a system call (e.g., reading a file) or otherwise in an inconsistent state
  - ✗ Can still leave data shared between user processes in an inconsistent state

Fall 2002, Lecture 18

#### **Round-Robin**

#### ■ Policy:

- Define a fixed time slice (also called a time quantum)
- Choose process from head of ready queue
- Run that process for <u>at most</u> one time slice, and if it hasn't completed by then, add it to the tail of the ready queue
- If that process terminates or blocks before its time slice is up, choose another process from the head of the ready queue, and run that process for at most one time slice...
- Implement using:
  - Hardware timer that interrupts at periodic intervals
  - FIFO ready queue (add to tail, take from head)

Fall 2002, Lecture 18

#### ■ Example 1:

Process (Arrival Order)	P1	P2	Р3
Burst Time	24	3	3
Arrival Time	0	0	0

**Round-Robin Example** 

 P1
 P2
 P3
 P1
 P1
 P1
 P1
 P1
 P1

 0
 4
 7
 10
 14
 18
 22
 26
 30

average waiting time = (4 + 7 + (10-4)) / 3 = 5.66

#### ■ Example 2:

Process (Arrival Order)	P3	P2	P1
Burst Time	3	3	24
Arrival Time	0	0	0

 P3
 P2
 P1
 P1
 P1
 P1
 P1
 P1
 P1

 0
 3
 6
 10
 14
 18
 22
 26
 30

average waiting time = (0 + 3 + 6) / 3 = 3

Fall 2002, Lecture 18

#### **Round-Robin Evaluation**

- Preemptive (at end of time slice)
- Response time good for short processes
  - Long processes may have to wait n\*q time units for another time slice
    - n = number of other processes,q = length of time slice
- Throughput depends on time slice
  - Too small too many context switches
  - ◆ Too large approximates FCFS
- Fairness penalizes I/O-bound processes (may not use full time slice)
- Starvation not possible
- Overhead low

## Shortest-Job-First (SJF)

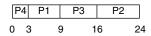
- Other names:
  - Shortest-Process-Next (SPN)
- Policy:
  - Choose the process that has the smallest next CPU burst, and run that process non-preemptively (until termination or blocking)
  - In case of a tie, FCFS is used to break the tie
- Difficulty: determining length of next CPU burst
  - Approximation predict length, based on past performance of the process, and on past predictions

Fall 2002, Lecture 18 8 Fall 2002, Lecture 18

### SJF Example

# ■ SJF Example:

Process (Arrival Order)	P1	P2	P3	P4
Burst Time	6	8	7	3
Arrival Time	0	0	0	0



average waiting time = (0 + 3 + 9 + 16) / 4 = 7

# ■ Same Example, FCFS Schedule:

average waiting time = (0 + 6 + 14 + 21) / 4 = 10.25

#### **SJF Evaluation**

- Non-preemptive
- Response time good for short processes
  - Long processes may have to wait until a large number of short processes finish
  - Provably optimal minimizes average waiting time for a given set of processes
- Throughput high
- Fairness penalizes long processes
- Starvation possible for long processes
- Overhead can be high (recording and estimating CPU burst times)

Fall 2002, Lecture 18 10 Fall 2002, Lecture 18