Announcements

- **Program #1**
  - Due Sunday

- **Reading**
  - Continue scheduling
Priority Based Scheduling

- **Priorities**
  - assign each process a priority, and scheduler always chooses process of higher priority over one of lower priority
- **More than one ready queue, ordered by priorities**

![Diagram of priority-based scheduling with queues and events](image-url)
Priority Algorithms

- **Fixed Queues**
  - processes are statically assigned to a queue
  - sample queues: system, foreground, background

- **Multilevel Feedback**
  - processes are dynamically assigned to queues
  - penalize jobs that have been running longer
  - preemptive, with dynamic priority
  - have $N$ ready queues (RQ0-RQN),
    - start process in RQ0
    - if quantum expires, moved to i + 1 queue
Feedback scheduling (cont.)

- problem: turnaround time for longer processes
  - can increase greatly, even starve them, if new short jobs regularly enter system
- solution1: vary preemption times according to queue
  - processes in lower priority queues have longer time slices
- solution2: promote a process to higher priority queue
  - after it spends a certain amount of time waiting for service in its current queue, it moves up
- solution3: allocate fixed share of CPU time to jobs
  - if a process doesn’t use its share, give it to other processes
  - variation on this idea: lottery scheduling
    - assign a process “tickets” (# of tickets is share)
    - pick random number and run the process with the winning ticket.
UNIX System V

- **Multilevel feedback, with**
  - RR within each priority queue
  - 10ms second preemption
  - priority based on process type and execution history, lower value is higher priority
- **priority recomputed once per second, and scheduler selects new process to run**
- **For process j, \( P(i) = \text{Base} + \frac{\text{CPU}(i-1)}{2} + \text{nice} \)**
  - \( P(i) \) is priority of process \( j \) at interval \( i \)
  - Base is base priority of process \( j \)
  - \( \text{CPU}(i) = \frac{\text{U}(i)}{2} + \frac{\text{CPU}(i-1)}{2} \)
    - \( \text{U}(i) \) is CPU use of process \( j \) in interval \( i \)
    - exponentially weighted average CPU use of process \( j \) through interval \( i \)
  - nice is user-controllable adjustment factor
UNIX (cont.)

- Base priority divides all processes into (non-overlapping) fixed bands of decreasing priority levels
  - swapper, block I/O device control, file manipulation, character I/O device control, user processes
- bands optimize access to block devices (disk), allow OS to respond quickly to system calls
- penalizes CPU-bound processes w.r.t. I/O bound
- targets general-purpose time sharing environment
Windows NT

● **Target:**
  – single user, in highly interactive environment
  – a server
● preemptive scheduler with multiple priority levels
● flexible system of priorities, RR within each, plus dynamic variation on basis of current thread activity for *some* levels
● 2 priority bands, real-time and variable, each with 16 levels
  – real-time ones have higher priority, since require immediate attention(e.g. communication, real-time task)
Windows NT (cont.)

- In real-time class, all threads have fixed priority that never changes
- In variable class, priority begins at an initial value, and can change, up or down
  - FIFO queue at each level, but thread can switch queues
- Dynamic priority for a thread can be from 2 to 15
  - if thread interrupted because time slice is up, priority lowered
  - if interrupted to wait on I/O event, priority raised
  - favors I/O-bound over CPU-bound threads
  - for I/O bound threads, priority raised more for interactive waits (e.g. keyboard, display) than for other I/O (e.g. disk)
Multi-Processor Scheduling

- **Multiple processes need to be scheduled together**
  - Called gang-scheduling
  - Allowing communicating processes to interact w/o waiting

- **Try to schedule processes back to same processor**
  - Called affinity scheduling
    - Maintain a small ready queue per processor
    - Go to global queue if nothing local is ready
Cooperating Processes

- Often need to share information between processes
  - information: a shared file
  - computational speedup:
    - break the problem into several tasks that can be run on different processors
    - requires several processors to actually get speedup
  - modularity: separate processes for different functions
    - compiler driver, compiler, assembler, linker
  - convenience:
    - editing, printing, and compiling all at once
Interprocess Communication

- Communicating processes establish a link
  - can more than two processes use a link?
  - are links one way or two way?
  - how to establish a link
    - how do processes name other processes to talk to
      - use the process id (signals work this way)
      - use a name in the filesystem (UNIX domain sockets)
      - indirectly via mailboxes (a separate object)

- Use send/receive functions to communicate
  - send(dest, message)
  - receive(dest, message)
Producer-consumer pair

- producer creates data and sends it to the consumer
- consumer read the data and uses it
- examples: compiler and assembler can be used as a producer consumer pair

Buffering
  - processes may not produce and consume items one by one
  - need a place to store produced items for the consumer
    - called a buffer
  - could be fixed size (bounded buffer) or unlimited (unbounded buffer)