• Scheduling
  o Introduction
    • Multiple threads or processes compete for the CPU simultaneously
      • Need to schedule them to achieve some goal
    • Process switching is expensive
      • Needs to use an algorithm that uses CPU efficiently
  o Process behavior is an important factor in scheduling
    • Almost all processes alternate between I/O and CPU calculations
    • Compute-bound (or CPU bound) – Process that spends most of its time computing
    • I/O bound – Process that spends most of its time waiting for I/O
      ▪ Takes the same amount of time to issue the I/O request no matter how long processing the data takes
      ▪ General principle – If an I/O bound process is ready, give it the CPU quickly so that it can issue its next I/O request
        • Will keep both the CPU and the disk busy
      ▪ As CPUs become faster, I/O bound processes are much more likely
        • Disk speed isn’t increasing as fast as CPU speed
  o When do we make scheduling decisions?
    • Process created/exits
    • Process blocks
    • I/O interrupt – process that is now ready, current process, or some other
    • Clock interrupt – A hardware clock that gives periodic interrupts
      • Non-preemptive – Pick a process and let it run indefinitely
        o Only releases when blocked or volunteers
        o When clock interrupt happens, the last process is always picked unless higher priority one exists(simple)
      • Preemptive – pick a process for a maximum of some fixed time
If still running at clock interrupt, suspended and the scheduler picks another process
Requires clock interrupt to give control to the scheduler

- Goals of scheduling algorithms
  - Generic
    - Fairness – give each process a fair share of the CPU
      - Fairness can be defined in many different ways
      - General rule – comparable processes should give comparable service
    - Policy enforcement – carrying out some stated policy
      - Example: Virus scanner code can run whenever it wants to
    - Balance – try to keep all parts of the system busy
      - If CPU and all devices are kept busy, more work is being done per second than if some are idle
      - If there is a mix of CPU and I/O bound processes, several can be “doing work” if the I/O bound processes are allowed to make their requests so they can be blocked.
        - Initially starving the I/O bound processes in favor of CPU bound will:
          - Increase contention for the CPU
          - Render the disks mostly idle
  - Three types of scheduling algorithms have different goals as well
  - Batch – not interactive with users waiting
    - Introduction
      - Payroll, inventory, etc.
      - Allows for either non-preemptive or very long access to CPU
      - Generic algorithms they use are good examples
    - Maximize throughput – number of jobs per hour that the system completes
    - Minimize turnaround time – average time a batch job takes to complete
      - Time it is submitted until moment completed
    - Maximizing throughput doesn’t necessarily minimize turnaround time
      - With mix of short and long jobs, schedule only the shortest jobs
        - Great throughput, but terrible turnaround
  - Interactive – users waiting for output with quick response
    - Introduction
      - Requires preemption to keep one process from hogging CPU
      - Servers that serve multiple remote users also fall under this category
- Minimize response time – amount of time between issuing a command (clicking a button) and getting the response
- Proportionality – Meet user expectations about how long something takes
  - Be consistent with response time for particular applications
  - Wild fluctuations tend to annoy users
    - Example: Firefox opens in 1 sec normally, but takes 30 seconds occasionally.
- Real time – Generally limited to specific job and doesn’t need preemption
  - Must meet deadlines – avoid losing data or causing jitter (multimedia)
  - We mostly won’t discuss real-time scheduling algorithms
- Batch scheduling
  - First-come first-served
    - Processes assigned to the CPU in the order they request it
    - Benefits
      - Requires a single queue for implementation
      - Very simple to code/understand
    - Negatives
      - Potentially inefficient use of resources
      - Example:
        - 1 CPU-bound process that reads for 1 sec and then does a disk request
        - Several I/O bound processes that require 1000 disk reads to complete
        - When CPU-bound process makes disk request, all I/O bound processes will make one disk request
        - Each I/O process will take 1000 seconds to complete
        - With preemption every 10ms, it would take 10 seconds each
  - Shortest Job First – assumes run times are known in advance
    - Realistic in the case of operations that are run very frequently
    - Several jobs are sitting in the queue waiting to be chosen

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

- Turnaround time: A = 8, B = 12, C = 16, D = 20
  - Average of 14 minutes
  - Shortest Job First rearranges them
- B = 4, C = 8, D = 12, A = 20
  - Average of 11 minutes
- Mean turnaround is \((4a+3b+2c+d)/4\)
  - Provably optimal when they all arrive simultaneously
  - Doesn’t hold up if they don’t
- Benefits: simple and can be provably optimal
- Negatives: requires knowing run time in advance, can’t be used preemptively
  - Shortest Remaining Time Next – preemptive version of shortest job first
    - Still requires knowing the running time
    - Scheduler always chooses the process whose remaining time is the shortest
    - Benefit: Allows new short jobs to get good service
- Interactive System Scheduling – servers, PCs, and general purpose systems
  - Round-robin – one of the oldest/simplest used
    - Introduction
      - Each process is assigned a time interval to run (quantum)
      - If process is running at end of the quantum, CPU is preempted and given to another process
      - If blocked, scheduling occurs then
    - Implementation – linked list of runnable processes

\[ \begin{array}{cccc}
  B & C & D & A \\
  4 & 4 & 4 & 8 \\
\end{array} \]

- After B uses its quantum

\[ \begin{array}{cccc}
  F & D & G & A \\
  B & & & \\
\end{array} \]

- Length of quantum is important
  - Process (context) switch takes quite awhile
    - Example: 4 msec quantum, 1 msec context switch
    - 20% waste on CPU overhead
• Too large is also a problem for servers
  o 100 msec quantum and 50 processes incoming
  o Last one might have a 5 second delay on processing
• Too large also means preemption doesn’t happen often
  o Longer than the mean CPU burst

**Conclusion:**
• Too short lowers CPU efficiency
• Too long hurts response time for short interactive processes
• Use some kind of compromise – 20-50msec