CPU Scheduling Questions

- Why is scheduling needed?
- What is preemptive scheduling?
- What are scheduling criteria?
- What are disadvantages and advantages of different scheduling policies, including:
  - First-come-first-serve?
  - Shortest job first?
  - Shortest time to completion first?
  - Round Robin?
  - Priority based?

Resources

- **Resource**: Anything that can be used by only a single process at any instant in time
- **Hardware device or a piece of information**
  - Examples:
    - CPU (time), Tape drive, Disk space, Memory
    - Locked record in a database (information)
- **Fungible resources**
  - Several interchangeable copies of a resource
    - Gold is fungible, one gram of gold is as good as any other
- **Focus today managing the CPU**

Resource Classification

- **Preemptable**
  - Characteristics:
    - small state (so it is not costly too preempt it).
    - only one resource
  - Examples:
    - CPU or Memory are typically a preemptable resources
- **Non-preemptable**
  - Cannot take a resource away from its current ‘owner’ without causing the computation to fail.

Resources Management Tasks

- **Allocation**:
  - **Space Sharing**: Which process gets which resource (control access to resource)?
- **Scheduling**:
  - **Time Sharing**: Which order should requests be serviced; Which process gets resource and at what time (order and time).


**Levels of CPU Managements**

- **Dispatcher** (low level mechanism)
  - Context Switch
    - Save execution of old process in PCB
    - Add PCB to appropriate queue (ready or blocked)
    - Load state of next process from PCB to registers
    - Switch from kernel to user mode
    - Jump to instruction in user process
  - Scheduler (higher level mechanism)
    - Policy to determine which process gets CPU when
  - Allocator
    - Policy to determine which processes compete for which CPU
    - Needed for multiprocessor, parallel, and distributed systems

**CPU Workload Model**

- Workload contains collection of jobs (processes)
- Job model
  - Job alternates between CPU usage and waiting for I/O
  - CPU-bound job:
    - Spends most of its time computing
    - Characteristics: Long CPU bursts and infrequent I/O waits
  - I/O-bound job:
    - Spends most of its time waiting for I/O
    - Characteristics: Short CPU bursts and frequent I/O waits
  - Trend: as CPUs get faster processes tend to get more I/O bound? (Why?)
- Do not know type of job before it executes
- Do not know duration of CPU or I/O burst
- Need job scheduling for each ready job
- Schedule each CPU burst

**I/O and CPU Bond Processes**

- Key factor is CPU bursts not the length of the I/O bursts
  - I/O boundless determine if they don’t compute much between I/O requests not because they have long I/O requests.

**Impact of Scheduling**

- Schedule another waiting process while current CPU relinquish to CPU due to I/O.

**Dispatch Mechanism (Review)**

*Dispatcher is the module that gives control of the CPU to the process selected by the scheduler.*

- OS runs dispatch loop:
  ```c
  while( forever )
  {
    run process A for some time slice
    stop process A and save it’s context
    load context of another process B
    jump to proper location and restart program
  }
  ```
  - How does the dispatcher gain control?
**Entering System Mode (Review)**

How does OS get control?
- Synchronous interrupts, or traps
  - Event internal to a process that gives control to OS
  - Examples: System calls, page faults (access page not in main memory), or errors (illegal instruction or divide by zero)
- Asynchronous interrupts
  - Events external to a process, generated by hardware
  - Examples: Characters typed, or completion of a disk transfer

How are interrupts handled?
- Each type of interrupt has corresponding routine (handler or interrupt service routine (ISR))
- Hardware saves current process and passes control to ISR

**How does Dispatcher run? (Review)**

Option 2: True Multi-tasking
- Trust process to relinquish CPU through traps
  - Trap: Event internal to process that gives control to OS
  - Examples: System call, page fault (access page not in main memory), or error (illegal instruction or divide by zero)
- Disadvantages: Processes can misbehave
  - By avoiding all traps and performing no I/O, can take over entire machine
  - Only solution: Reboot!
- Not performed in modern operating systems

**Scheduling Performance Metrics**

- **Trade-off between maximizing:**
  - System’s point of view: Overall efficiency
  - User’s point of view: Giving good service to individual processes

**Overall Efficiency**

- **System Load (uptime):**
  - The amount of work the system is doing
- **Throughput:**
  - Want many jobs to complete per unit time
- **System Utilization:**
  - Keep expensive devices busy
  - Jobs arrive infrequently and both throughput and system utilization is low
- **Example:** Lightly loaded system - jobs arrive infrequently - both throughput and system utilization is low.
- **Scheduling Goal:** Ensure that throughput increase linearly with load
Good Service

- Ensure that processes quickly start, run and completes.
- **Turnaround time**: The time between job arrival and job completion
  - **Response time**: The length of time when the job arrive and when if first start to produce output
    - e.g. interactive jobs, virtual reality (VR) games, click on mouse see VR change
  - **Waiting time**: Time in ready queue - do not want to spend a lot of time in the ready queue
    - Better ‘scheduling’ quality metric than turn-around time since scheduler does not have control over blocking time or time a process does actual computing.
- **Fairness**: all jobs get the same amount of CPU over time
- **Overhead**: reduce number of context switches
- **Penalty Ratio**: Elapsed time / Service time

Penalty Ratio

- Comparison to an ideal system: How much time worse turn-around time would compared to an ideal system that would only consist of ‘service time’.
- Lower penalty ratio is better
- Examples:
  - 1 indicates ‘no’ penalty (job never waits)
  - 2 indicates it takes twice as long than an ideal system.

Criteria Depends on System

- All Systems:
  - Fairness
  - Overall system utilization
  - Policy enforcement (priorities)
- Batch Systems (not interactive)
  - Throughput
  - Turn-around time
  - CPU utilization
- Real-time system (real time constraints)
  - Meeting deadlines (avoid losing data)
  - Predictability - avoid quality degradation in multimedia systems.

Gantt Chart

- Shows how jobs are scheduled over time on the CPU
- Job: Arrival | CPU burst | Time
  - A: 0 | 10 | 0
  - B: 1 | 2 | 10
  - C: 2 | 4 | 12

First-Come-First-Served (FCFS)

- **Idea**: Maintain FIFO list of jobs as they arrive
  - Non-preemptive policy
  - Allocate CPU to job at head of list

  **Example**:
  - 1 CPU bound job
  - 3 I/O bound jobs

**Penalty Chart**

- **Penalty ratio** = Total elapsed time / Service time: doing actual work (on CPU + doing I/O)

**FCFC Problem**

- Convoy effect -- if there is not a nice balance of I/O bound jobs and CPU bound jobs
  - Example:
    - 1 CPU bound job
    - 3 I/O bound jobs
Convoy Effect

- All I/O devices idle even when the system contains lots of I/O jobs
  - CPU-bound jobs get CPU and holds it
  - I/O-bound jobs move onto ready queue and waits

FCFS Discussion

- **Advantage:**
  - Simple implementation (less errors)
- **Disadvantages:**
  - Waiting time depends on arrival order
  - Potentially long jobs wait for jobs that arrive later
  - Tend to favor long bursts (CPU-bound processes)
    - Want to favor short bursts since they will finish quickly and not crowd the ready list.
  - Shortest burst
  - Convoy effect: Short jobs stuck waiting for long jobs
    - Hurt waiting time for short jobs
    - Reduces utilization of I/O devices

SJF Discussion

- **Advantages**
  - Provably optimal for minimizing average wait time (with no preemption)
  - Moving shorter job before longer job improves waiting time of short job more than it harms waiting time of long job
  - Helps keep I/O devices busy
- **Disadvantages**
  - Not practical: Cannot predict future CPU burst time
  - OS solution: Use past behavior to predict future behavior
- **Starvation:** Long jobs may never be scheduled
Shortest-Time-to-Completion-First (STCF/SCTF)

Idea: Add preemption to SJF
- Schedule newly ready job if shorter than remaining burst for running job

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival</th>
<th>CPU burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

STCF Average wait:

<table>
<thead>
<tr>
<th>SJF Average wait:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Round-Robin (RR)

Idea: Run each job for a time-slice and then move to back of FIFO queue
- Preempt job if still running at end of time-slice

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival</th>
<th>CPU burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Average wait:

<table>
<thead>
<tr>
<th>Average wait:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

RR Discussion

Advantages
- Jobs get fair share of CPU
- Shortest jobs finish relatively quickly

Disadvantages
- Poor average waiting time with similar job lengths
  - Example: 10 jobs each requiring 10 time slices
  - RR: All complete after about 100 time slices
  - FCFS performs better!
- Performance depends on length of time-slice
  - If time-slice too short, pay overhead of context switch
  - If time-slice too long, degenerate to FCFS

Priority Based

Idea: Each job is assigned a priority
- Schedule highest priority ready job
- May be preemptive or non-preemptive
- Priority may be static or dynamic

Advantages
- Static priorities work well for real time systems
- Dynamic priorities work well for general workloads

Disadvantages
- Low priority jobs can starve
- How to choose priority of each job?

Goal: Adjust priority of job to match CPU burst
- Approximate SCTF by giving short jobs high priority