CPU Scheduling

Sarah Diesburg
Operating Systems
CS 3430
CPU Scheduler

- A **CPU scheduler** is responsible for
  - Removal of running process from the CPU
  - Selection of the next running process
    - Based on a particular strategy
Goals for a Scheduler

- Maximize
  - \textit{CPU utilization}: keep the CPU as busy as possible
  - \textit{Throughput}: the number of processes completed per unit time
Goals for a Scheduler

- Minimize
  - **Response time**: the time of submission to the time the first response is produced
  - **Wait time**: total time spent waiting in the ready queue
  - **Turnaround time**: the time of submission to the time of completion
Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0.
- We want to know the response time, waiting time, and turnaround time of process A.

![Diagram showing processes A, B, and C over time]

- Turnaround time = Response time + Waiting time
- Response time = 0

Time:
A B C A B C A C A C
Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0
- We want to know the response time, waiting time, and turnaround time of process B

\[
\text{turnaround time} = \text{wait time} + \text{response time}
\]

Diagram:

- Time line with processes A, B, C, A, B, C, A, C, A, C
- Different colors or symbols for each process
Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0
- We want to know the response time, waiting time, and turnaround time of process C
Goals for a Scheduler

- Achieve *fairness*
  - There are tensions among these goals
Assumptions

- Each user runs one process
- Each process is single threaded
- Processes are independent

They are not realistic assumptions; they serve to simplify analyses
Scheduling Policies

- FIFO (first in, first out)
- Round robin
- SJF (shortest job first)
- Multilevel feedback queues
- Lottery scheduling
FIFO

- **FIFO**: assigns the CPU based on the order of requests
  - **Nonpreemptive**: A process keeps running on a CPU until it is blocked or terminated
  - Also known as FCFS (first come, first serve)
  + Simple
  - Short jobs can get stuck behind long jobs
Round Robin

- **Round Robin (RR)** periodically releases the CPU from long-running jobs
  - Based on timer interrupts so short jobs can get a fair share of CPU time
  - **Preemptive**: a process can be forced to leave its running state and replaced by another running process
  - **Time slice**: interval between timer interrupts
More on Round Robin

- If time slice is too long
  - Scheduling degrades to FIFO

- If time slice is too short
  - Throughput suffers
  - Context switching cost dominates
More on Round Robin

- Round robin based on FIFO
  - Gives time slice to process that has waited longest to run
  - Used to choose next process to run
More on Round Robin

- Suppose we have three jobs of equal length

  A B C A

  Round Robin

- Which job goes next?

  A B C A B

  Round Robin
FIFO vs. Round Robin

- With zero-cost context switch, is RR always better than FIFO?
FIFO vs. Round Robin

Suppose we have three jobs of equal length

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turnaround time of A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turnaround time of B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turnaround time of C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Round Robin

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turnaround time of A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turnaround time of B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turnaround time of C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FIFO vs. Round Robin

Round Robin
+ Shorter response time
+ Fair sharing of CPU
- Not all jobs are preemptive
- Not good for jobs of the same length
Shortest Job First (SJF)

- **SJF** runs whatever job puts the least demand on the CPU, also known as **STCF (shortest time to completion first)**
  + Provably optimal
  + Great for short jobs
  + Small degradation for long jobs

- Real life example: supermarket express checkouts
SJF Illustrated

- turnaround time of C
- turnaround time of B
- turnaround time of A
  - wait time of C
  - wait time of B
  - wait time of A = 0
- response time of C
- response time of B
- response time of A = 0

Shortest Job First
Shortest Remaining Time First (SRTF)

- **SRTF**: a preemptive version of SJF
  - If a job arrives with a shorter time to completion, SRTF preempts the CPU for the new job
  - Also known as **SRTCF (shortest remaining time to completion first)**
  - Generally used as the base case for comparisons
SJF and SRTF vs. FIFO and Round Robin

- If all jobs are the same length, SJF → FIFO
  - FIFO is the best you can do
- If jobs have varying length
  - Short jobs do not get stuck behind long jobs under SRTF
A More Complicated Scenario (Arrival Times = 0)

- Process A (6 units of CPU request)
  - 100% CPU
  - 0% I/O

- Process B (6 units of CPU request)
  - 100% CPU
  - 0% I/O

- Process C (infinite loop)
  - 33% CPU
  - 67% I/O
A More Complicated Scenario

- **FIFO**
  - **CPU**
  - **I/O**

  Poor response and wait time for process C

- **Round Robin with time slice = 3 units**
  - **CPU**
  - **I/O**

  Disk utilization: 29% (2 out of 7 units)
## A More Complicated Scenario

### Round Robin with time slice = 1 unit

- **CPU**
  - Processing sequence: A → B → C → A → B → C

- **I/O**
  - Processing sequence: C

Disk utilization: 66% (2 out of 3 units)

### SRTF

- **CPU**
  - Processing sequence: C → A → C → A → C → A → C → B → C → B

- **I/O**
  - Processing sequence: C

Disk utilization: 66% (2 out of 3 units)
**Drawbacks of Shortest Job First**

- **Starvation**: constant arrivals of short jobs can keep long ones from running

- There is no way to know the completion time of jobs (most of the time)

  - Some solutions
    - Ask the user, who may not know any better
    - If a user cheats, the job is killed
Priority Scheduling (Multilevel Queues)

- *Priority scheduling:* The process with the highest priority runs first

- Priority 0: C
- Priority 1: A
- Priority 2: B

- Assume that low numbers represent high priority
Priority Scheduling

+ Generalization of SJF
  - With SJF, priority = 1/requested_CPU_time
- Starvation
Multilevel Feedback Queues

- **Multilevel feedback queues** use multiple queues with different priorities
  - Round robin at each priority level
  - Run highest priority jobs first
  - Once those finish, run next highest priority, etc
  - Jobs start in the highest priority queue
  - If time slice expires, drop the job by one level
  - If time slice does not expire, push the job up by one level
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

![Diagram showing three priorities with time slices 1, 2, and 4, and tasks A, B, C at time 0.]

Time
Multilevel Feedback Queues

- Priority 0 (time slice = 1): B
- Priority 1 (time slice = 2): A
- Priority 2 (time slice = 4):
Multilevel Feedback Queues

- Priority 0 (time slice = 1): C
- Priority 1 (time slice = 2): A B
- Priority 2 (time slice = 4): A B

Time = 2
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

\[
\begin{align*}
&\text{Time} = 3 \\
&\text{A} \quad \text{B} \quad \text{C}
\end{align*}
\]
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

Suppose process A is blocked on an I/O
Multilevel Feedback Queues

- Priority 0 (time slice = 1): A
- Priority 1 (time slice = 2): B C
- Priority 2 (time slice = 4):

suppose process A is blocked on an I/O
Multilevel Feedback Queues

- Priority 0 (time slice = 1): A
- Priority 1 (time slice = 2): C
- Priority 2 (time slice = 4):

Suppose process A is returned from an I/O.
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

![Diagram showing task order and time slices]
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4): c
Multilevel Feedback Queues

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

```
+--------+--------+--------+--------+--------+--------+
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|  A     |  B     |  C     |  B     |  A     |  C     |
+--------+--------+--------+--------+--------+--------+
```

Time = 9
Multilevel Feedback Queues

- Approximates SRTF
  - A CPU-bound job drops like a rock
  - I/O-bound jobs stay near the top
  - Still unfair for long running jobs
  - Counter-measure: *Aging*
    - Increase the priority of long running jobs if they are not serviced for a period of time
    - Tricky to tune aging
Lottery Scheduling

- *Lottery scheduling* is an adaptive scheduling approach to address the fairness problem
  - Each process owns some tickets
  - On each time slice, a ticket is randomly picked
  - On average, the allocated CPU time is proportional to the number of tickets given to each job
Lottery Scheduling

- To approximate SJF, short jobs get more tickets
- To avoid starvation, each job gets at least one ticket
## Lottery Scheduling Example

- **short jobs:** 10 tickets each
- **long jobs:** 1 ticket each

<table>
<thead>
<tr>
<th># short jobs/# long jobs</th>
<th>% of CPU for each short job</th>
<th>% of CPU for each long job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>91%</td>
<td>9%</td>
</tr>
<tr>
<td>0/2</td>
<td>0%</td>
<td>50%</td>
</tr>
<tr>
<td>2/0</td>
<td>50%</td>
<td>0%</td>
</tr>
<tr>
<td>10/1</td>
<td>10%</td>
<td>1%</td>
</tr>
<tr>
<td>1/10</td>
<td>50%</td>
<td>5%</td>
</tr>
</tbody>
</table>