Last Class: Processes

- A process is the unit of execution.
- Processes are represented as Process Control Blocks in the OS
  - PCBs contain process state, scheduling and memory management information, etc
- A process is either New, Ready, Waiting, Running, or Terminated.
- On a uniprocessor, there is at most one running process at a time.
- The program currently executing on the CPU is changed by performing a context switch
- Processes communicate either with message passing or shared memory

Today: Scheduling Algorithms

- Goals for scheduling

- FCFS & Round Robin

- SJF

- Multilevel Feedback Queues

- Lottery Scheduling
Scheduling Processes

• **Multiprogramming**: running more than one process at a time enables the OS to increase system utilization and throughput by overlapping I/O and CPU activities.

• Process Execution State

![Process State Diagram]

• All of the processes that the OS is currently managing reside in one and only one of these state queues.

Scheduling Processes

• **Long Term Scheduling**: How does the OS determine the degree of multiprogramming, i.e., the number of jobs executing at once in the primary memory?

• Short Term Scheduling: How does (or should) the OS select a process from the ready queue to execute?
  – Policy Goals
  – Policy Options
  – Implementation considerations
Short Term Scheduling

• The kernel runs the scheduler at least when
  1. a process switches from running to waiting,
  2. an interrupt occurs, or
  3. a process is created or terminated.

• **Non-preemptive system**: the scheduler must wait for one of these events

• **Preemptive system**: the scheduler can interrupt a running process

Criteria for Comparing Scheduling Algorithms

• **CPU Utilization** The percentage of time that the CPU is busy.

• **Throughput** The number of processes completing in a unit of time.

• **Turnaround time** The length of time it takes to run a process from initialization to termination, including all the waiting time.

• **Waiting time** The total amount of time that a process is in the ready queue.

• **Response time** The time between when a process is ready to run and its next I/O request.
Scheduling Policies

Ideally, choose a CPU scheduler that optimizes all criteria simultaneously (utilization, throughput,..), but this is not generally possible.

Instead, choose a scheduling algorithm based on its ability to satisfy a policy:

- Minimize average response time - provide output to the user as quickly as possible and process their input as soon as it is received.
- Minimize variance of response time - in interactive systems, predictability may be more important than a low average with a high variance.
- Maximize throughput - two components
  - minimize overhead (OS overhead, context switching)
  - efficient use of system resources (CPU, I/O devices)
- Minimize waiting time - give each process the same amount of time on the processor. This might actually increase average response time.

Simplifying Assumptions

- One process per user
- One thread per process
- Processes are independent

Researchers developed these algorithms in the 70's when these assumptions were more realistic, and it is still an open problem how to relax these assumptions.
Scheduling Algorithms: A Snapshot

**FCFS:** First Come, First Served

**Round Robin:** Use a time slice and preemption to alternate jobs.

**SJF:** Shortest Job First

**Multilevel Feedback Queues:** Round robin on each priority queue.

**Lottery Scheduling:** Jobs get tickets and scheduler randomly picks winning ticket.

Scheduling Policies

**FCFS:** First-Come-First-Served (or FIFO: First-In-First-Out)

- The scheduler executes jobs to completion in arrival order.
- In early FCFS schedulers, the job did not relinquish the CPU even when it was doing I/O.
- We will assume a FCFS scheduler that runs when processes are blocked on I/O, but that is non-preemptive, i.e., the job keeps the CPU until it blocks (say on an I/O device).
FCFS Scheduling Policy: Example

```
arrival order: B,C,A (no I/O)

arrival order: A,B,C (no I/O)

arrival order: A,B,C (A does I/O)
```

- If processes arrive 1 time unit apart, what is the average wait time in these three cases?

**FCFS: Advantages and Disadvantages**

**Advantage:** simple

**Disadvantages:**
- average wait time is highly variable as short jobs may wait behind long jobs.
- may lead to poor overlap of I/O and CPU since CPU-bound processes will force I/O bound processes to wait for the CPU, leaving the I/O devices idle
Round Robin Scheduling

- Variants of round robin are used in most time sharing systems
- Add a timer and use a preemptive policy.
- After each time slice, move the running thread to the back of the queue.
- Selecting a time slice:
  - Too large - waiting time suffers, degenerates to FCFS if processes are never preempted.
  - Too small - throughput suffers because too much time is spent context switching.
  => Balance these tradeoffs by selecting a time slice where context switching is roughly 1% of the time slice.
- Today: typical time slice = 10-100 ms, context switch time = 0.1-1 ms
- **Advantage:** It's fair; each job gets an equal shot at the CPU.
- **Disadvantage:** Average waiting time can be bad.

### Round Robin Scheduling: Example 1

- 5 jobs, 100 seconds each, time slice 1 second, context switch time of 0

<table>
<thead>
<tr>
<th>Job</th>
<th>Length</th>
<th>Completion Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FCFS Round Robin</td>
<td>FCFS Round Robin</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Round Robin Scheduling: Example 1

- 5 jobs, 100 seconds each, time slice 1 second, context switch time of 0

<table>
<thead>
<tr>
<th>Job</th>
<th>Length</th>
<th>Completion Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FCFS</td>
<td>Round Robin</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>496</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>200</td>
<td>497</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>300</td>
<td>498</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>400</td>
<td>499</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>Average</td>
<td>300</td>
<td>498</td>
<td>200</td>
</tr>
</tbody>
</table>

### Round Robin Scheduling: Example 2

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

<table>
<thead>
<tr>
<th>Job</th>
<th>Length</th>
<th>Completion Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FCFS</td>
<td>Round Robin</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Round Robin Scheduling: Example 2

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

<table>
<thead>
<tr>
<th>Job</th>
<th>Length</th>
<th>Completion Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FCFS</td>
<td>Round Robin</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>50</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>90</td>
<td>140</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>140</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Average</td>
<td>110</td>
<td>110</td>
<td>80</td>
</tr>
</tbody>
</table>

SJF/SRTF: Shortest Job First

- Schedule the job that has the least (expected) amount of work (CPU time) to do until its next I/O request or termination.

- **Advantages:**
  - Provably optimal with respect to minimizing the average waiting time
  - Works for preemptive and non-preemptive schedulers
  - Preemptive SJF is called SRTF - shortest remaining time first

  $\Rightarrow$ I/O bound jobs get priority over CPU bound jobs

- **Disadvantages:**
  - Impossible to predict the amount of CPU time a job has left
  - Long running CPU bound jobs can starve
**SJF: Example**

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

<table>
<thead>
<tr>
<th>Job</th>
<th>Length</th>
<th>Completion Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FCFS RR SJF FCFS RR SJF</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>50</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>90</td>
<td>140</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>140</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>110</td>
<td>110</td>
</tr>
</tbody>
</table>