Outline

1. Recall
2. Basics
   - Concepts
   - Criteria
3. Algorithms
4. Multi-Processor Scheduling
- Interrupts
- Traps (software errors, illegal instructions)
- System calls
Recall Basics Algorithms Multi-Processor Scheduling

**PCB**
- process state
- process ID (number)
- PC
- Registers
- memory information
- open files
- : other resources

**Job Queue**
- Linked list of PCBs
  - (main) job queue
  - ready queue
  - device queues

**Schedulers**
- Long-term/Job scheduler
  (loads from disk)
- Short-term/CPU scheduler
  (dispatches from ready queue)
Note that...

On Operating Systems which support threads, it is kernel-level threads – *not processes* – that are being scheduled.

However, *process* scheduling $\approx$ *thread* scheduling.
CPU and IO Bursts

- load, store, add, store, read from file
  - Wait for IO
- store, increment, branch, write to file
  - Wait for IO
- load, store, read from file
  - Wait for IO

**CPU Burst cycles**
Intervals with no I/O usage

**Waiting time**
Sum of time waiting in ready queue
When should we schedule a process?

- From *running* state to *waiting* state
- From *running* state to *ready* state
- From *waiting* state to *ready* state
- Terminates

**Scheme**

- non-preemptive or cooperative

**Scheme**

- preemptive
How do we select the next process?

- **CPU utilization**
  - CPU as busy as possible

- **Throughput**
  - Number of processes that are completed per time unit

- **Turnaround time**
  - Time between submission and completion

- **Waiting time**
  - Scheduling affects only waiting time

- **Response time**
  - Time between submission and first response
First Come, First Served (FCFS)

- Non-preemptive
- Treats ready queue as FIFO.
- Simple, but typically long/varying waiting time.
First Come, First Served (FCFS)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Gantt chart: Order $P_1$, $P_2$, $P_3$

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average waiting time: $(0+24+27)/3 = 17$
First Come, First Served (FCFS)

**Example**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

**Gantt chart: Order $P_2$, $P_3$, $P_1$**

<table>
<thead>
<tr>
<th></th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_1$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>

Average waiting time: $(0+3+6)/3 = 3$
Convoy effect

Consider:

- $P_1$: CPU-bound
- $P_2$, $P_3$, $P_4$: I/O-bound
Convoy effect

- $P_2$, $P_3$ and $P_4$ could quickly finish their IO request $\Rightarrow$ ready queue, waiting for CPU.
- Note: IO devices are idle then.
- then $P_1$ finishes its CPU burst and move to an IO device.
- $P_2$, $P_3$, $P_4$, which have short CPU bursts, finish quickly $\Rightarrow$ back to IO queue.
- Note: CPU is idle then.
- $P_1$ moves then back to ready queue is gets allocated CPU time.
- Again $P_2$, $P_3$, $P_4$ wait behind $P_1$ when they request CPU time.

One cause: **FCFS is non-preemptive**

$P_1$ keeps the CPU as long as it needs
Shortest Job First (SJF)

- Give CPU to the process with the shortest next burst
- If equal, use FCFS
- Better name: shortest next cpu burst first

Assumption

Know the length of the next CPU burst of each process in Ready Queue
Short Job First (SJF)

Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Gantt chart: Order $P_1$, $P_2$, $P_3$, $P_4$

<table>
<thead>
<tr>
<th></th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_2$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>9</td>
<td>16</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

Average waiting time: $(0+3+16+9)/4 = 7$
With FCFS: $(0+6+(6+8)+(6+8+7))/4 = 10.25$
SJF – Characteristics

Optimal wrt. waiting time!

Problem: how to know the next burst?

- User specifies (e.g. for batch system)
- Guess/predict based on earlier bursts, using exponential average:
  \[ \tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n \]
- \( t_n \): most recent information
- \( \tau_n \): past history

Can be preemptive or not
SJF with Preemption

Shortest Remaining Time First

When a process arrives to RQ, sort it in and select the SJF including the running process, possibly interrupting it.

(Remember: SJF schedules a new process only when the running is finished)
SJF with Preemption

**Example**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

**Gantt chart**

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>17</td>
<td>26</td>
</tr>
</tbody>
</table>

Average waiting time: 
\[
\frac{(10-1)+(1-1)+(17-2)+(5-3))}{4} = 6.5
\]

With SJF: 
\[
\frac{(0+4+(4+5)+(4+5+8))}{4} = 7.75
\]
Priority Scheduling Algorithms

- Priority associated with each process
- CPU allocated to the process with highest priority
- If equal, use FCFS

Note: SJF is a priority scheduling algorithm with

\[ p = \frac{1}{(predicted\ next\ CPU\ burst)} \]
### Priority Scheduling Algorithms

#### Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>10</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>$P_5$</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Gantt chart**

<table>
<thead>
<tr>
<th></th>
<th>$P_2$</th>
<th>$P_5$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>16</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

**Average waiting time:** $(0+1+6+16+18)/5 = 8.2$
Priority Criteria

- **Internal Priority**
  - time limits, mem requirements, number of open files, ratio $\frac{\text{Average IO burst}}{\text{Average CPU burst}}$

- **External Priority**
  - Criteria outside the OS. Choice related to computer usage.

- Can be preemptive or not

- Problem: **Starvation** (or Indefinite Blocking)

- Solution: **Aging**
Round-Robin (RR)

- FCFS with Preemption
- **Time quantum** (or time slice)
- Ready Queue treated as circular queue
Round-Robin (RR)

Example

Quantum $q = 4$

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Gantt chart

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_1$</th>
<th>…</th>
<th>$P_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>14</td>
<td>26</td>
</tr>
</tbody>
</table>

Average waiting time: $(0+4+7+(10-4))/3 = 5.66$
With FCFS: $(0+24+27)/3 = 17$
RR – Characteristics

- Turnaround time typically larger than SRTF but better response time

- Performance depends on quantum $q$
  - Small $q$: Overhead due to context switches (& scheduling)
    - $q$ should be large wrt context-switching time
  - Large $q$: Behaves like FCFS
    - rule of thumb: 80% of bursts should be shorter than $q$ (also improves turnaround time)
Multilevel Queue Scheduling

Observation
Different algorithms suit different types of processes (e.g. interactive vs batch/background processes) and systems are often not only running interactive or "batch" processes.

Multilevel queues
We split the Ready Queue in several queues, each with its own scheduling algorithm

Example
interactive processes: RR
background processes: FCFS/SRTF
Multilevel Queue – Scheduling among Queues

One more dimension
We need scheduling between the Ready Queues

Example (Common implementation)
Fixed-priority preemption (with priority to interactive processes)
Multilevel Queue – More complex example

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes

where each queue has absolute priority over lower-priority queues.

No process in low-priority queues can run if high-priority queues are not empty.

So, if a lower-priority queue is only used when all higher-priority RQs are empty & higher-priority processes preempt lower-priority ones, we risk starvation.

Possible solution: give time-slices to each Ready Queue
(basically RR between the queues, with different quanta for each queue)

⇒ Each queue gets a certain guaranteed slice of the CPU time.
Multi-Level Feedback Queue Scheduling (MLFQ)

With MLQ, each process is permanently assigned to one queue (based on type, priority etc).

**MLFQ**

allow processes to move between queues

Idea: Separate processes according to their CPU bursts.

**Example**

- Let processes with long CPU bursts move down in the queue levels
- Leave I/O bound and interactive processes in high-priority queues
- Combine with aging principle to prevent starvation
MLFQ – Example

1. Round-Robin with quantum 8
2. Round-Robin with quantum 16
3. FCFS

$Q_i$ has priority over, and preempts, $Q_{i+1}$. New processes are added to $Q_1$. If a process in $Q_1$ or $Q_2$ does not finish within its quantum, it is moved down to the next queue. Thus:

- short bursts (I/O bound and interactive proc) are served quickly;
- slightly longer are also served quickly but with less priority;
- long (CPU bound processes) are served when there is CPU to be spared.
Symmetry / Asymmetry

Asymmetric MPs scheduling
One Master Server does all scheduling.
Others execute only user code

Symmetric MPs (SMP) scheduling
Each processor does scheduling.
(whether CPUs have a common or private Ready Queues)
Try to keep a process on the same processor as last time, because of **Geographical Locality**

(Moving the process to another CPU causes cache misses)

- **Soft affinity**
  - The process *may move* to another processor

- **Hard affinity**
  - The process *must stay* on the same processor
Load Balancing

Keep the workload evenly distributed over the processors

- **push migration**
  periodically check the load, and "push" processes to less loaded queues.

- **pull migration**
  idle processors "pull" processes from busy processors

Note: Load balancing goes against processor affinity.
Hyperthreaded CPUs

CPUs with multiple "cores"

Sharing cache and bus influences affinity concept and thus scheduling.

The OS can view each core as a CPU, but can make additional benefits with threads.