Scheduling I

- Introduction to scheduling
- Scheduling algorithms
Role of Dispatcher vs. Scheduler

- **Dispatcher**
  - Low-level mechanism
  - Responsibility: context switch

- **Scheduler**
  - High-level policy
  - Responsibility: deciding which process to run

- Could have an allocator for CPU as well
  - Parallel and distributed systems
When to schedule?

- When does scheduler make decisions?
  - When a process
    1. switches from running to waiting state
    2. switches from running to ready state
    3. switches from waiting to ready
    4. terminates

- Minimal: nonpreemptive
  - ?

- Additional circumstances: preemptive
  - ?
Overview of scheduling algorithms

- **Criteria:** workload and environment

- **Workload**
  - Process behavior: alternating sequence of CPU and I/O bursts
  - CPU bound v.s. I/O bound

- **Environment**
  - Batch v.s. interactive?
  - Specialized v.s. general?
Scheduling performance metrics

- **Min waiting time**: don’t have process wait long in ready queue
- **Max CPU utilization**: keep CPU busy
- **Max throughput**: complete as many processes as possible per unit time
- **Min response time**: respond immediately
- **Fairness**: give each process (or user) same percentage of CPU
First-Come, First-Served (FCFS)

- Simplest CPU scheduling algorithm
  - First job that requests the CPU gets the CPU
  - Nonpreemptive

- Implementation: FIFO queue
Example of FCFS

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Gantt chart**

Schedule: 

- $P_1$
- $P_2$
- $P_3$
- $P_4$

- **Average waiting time:** $(0 + 7 + 11 + 12)/4 = 7.5$
### Example of FCFS: different arrival order

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Arrival order: P₃  P₂  P₄  P₁

- Average waiting time: \((9 + 1 + 0 + 5)/4 = 3.75\)
FCFS advantages and disadvantages

- **Advantages**
  - Simple
  - Fair

- **Disadvantages**
  - waiting time depends on arrival order
  - *Convoy effect*: short process stuck waiting for long process
  - Also called head of the line blocking
Shortest Job First (SJF)

- Schedule the process with the shortest time
- FCFS if same time
Example of SJF (w/o preemption)

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Gantt chart

Schedule: $P_1$ $P_3$ $P_2$ $P_4$

Arrival: $P_1$ $P_2$ $P_3$ $P_4$

- Average waiting time: $(0 + 6 + 3 + 7)/4 = 4$
SJF Advantages and Disadvantages

- **Advantages**
  - Minimizes average wait time. Provably optimal if no preemption allowed

- **Disadvantages**
  - Not practical: difficult to predict burst time
    - Possible: past predicts future
  - May starve long jobs
Shortest Remaining Time First (SRTF)

- If new process arrives with shorter CPU burst than the remaining for current process, schedule new process
  - SJF with preemption

- Advantage: reduces average waiting time
  - Provably optimal
Example of SRTF

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Gantt chart**

Schedule: P₁ P₂ P₃ P₂ P₄ P₁

Arrival: P₁ P₂ P₃ P₄

- **Average waiting time:** \( \frac{9 + 1 + 0 + 2}{4} = 3 \)
Round-Robin (RR)

- Practical approach to support time-sharing
- Run process for a time slice, then move to back of FIFO queue
- Preempted if still running at end of time-slice
- How to determine time slice?
Example of RR: time slice = 3

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Average waiting time: \( \frac{8 + 8 + 5 + 7}{4} = 7 \)
- Average response time: \( \frac{0 + 1 + 5 + 5}{4} = 2.75 \)
- # of context switches: 7
**Smaller time slice = 1**

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_1)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>(P_2)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(P_3)</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>(P_4)</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Average waiting time: \((8 + 6 + 1 + 7)/4 = 5.5\)
- Average response time: \((0 + 0 + 1 + 2)/4 = 0.75\)
- \# of context switches: 14
Larger time slice = 10

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Average waiting time: \(\frac{0 + 5 + 7 + 7}{4} = 4.75\)
- Average response time: same
- # of context switches: 3 (minimum)
RR advantages and disadvantages

- **Advantages**
  - Low response time, good interactivity
  - Fair allocation of CPU across processes
  - Low average waiting time when job lengths vary widely

- **Disadvantages**
  - Poor average waiting time when jobs have similar lengths
    - Average waiting time is even worse than FCFS!
  - Performance depends on length of time slice
    - Too high \(\Rightarrow\) degenerate to FCFS
    - Too low \(\Rightarrow\) too many context switches, costly
Priorities

- A priority is associated with each process
  - Run highest priority ready job (some may be blocked)
  - Round-robin among processes of equal priority
  - Can be preemptive or non-preemptive

- Representing priorities
  - Typically an integer
  - The larger the higher or the lower?
Setting priorities

- Priority can be statically assigned
  - Some always have higher priority than others
  - Problem: starvation

- Priority can be dynamically changed by OS
  - Aging: increase the priority of processes that wait in the ready queue for a long time

```c
for(pp = proc; pp < proc+NPROC; pp++) {
    if (pp->prio != MAX)
        pp->prio++;
    if (pp->prio > curproc->prio)
        reschedule();
}
```

This code is taken almost verbatim from 6th Edition Unix, circa 1976.
Priority inversion

- High priority process depends on low priority process (e.g. to release a lock)
  - Another process with in-between priority arrives?

```
P1 (low): lock(my_lock) (gets my_lock)
P2(high): lock(my_lock)
P3(medium): while (...) {} 
P2 waits, P3 runs, P1 waits
P2's effective priority less than P3!
```

- **Solution**: priority inheritance
  - Inherit highest priority of waiting process
  - Must be able to chain multiple inheritances
  - Must ensure that priority reverts to original value

- **Google for “mars pathfinder priority inversion”**
Backup slides
Multilevel Queue

- Ready queue is partitioned into separate queues: foreground (interactive) and background (batch).
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS
Multilevel Queue Scheduling

- Highest priority: system processes
- Interactive processes
- Interactive editing processes
- Batch processes
- Student processes

Lowest priority
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service
Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – RR with time quantum 8 milliseconds
  - $Q_1$ – RR time quantum 16 milliseconds
  - $Q_2$ – FCFS

- Scheduling
  - A new job enters queue $Q_0$ which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue $Q_1$.
  - At $Q_1$ job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$. 
Multilevel Feedback Queues

- quantum = 8
- quantum = 16
- FCFS
Example of RR with Time Slice = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>17</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

- The Gantt chart is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>37</td>
<td>57</td>
<td>77</td>
<td>97</td>
<td>117</td>
<td>121</td>
<td>134</td>
<td>154</td>
<td>162</td>
</tr>
</tbody>
</table>

- Typically, higher average turnaround than SJF, but better *response*
Example of RR with Time Slice = 20

<table>
<thead>
<tr>
<th></th>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P_1</strong></td>
<td>0</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td><strong>P_2</strong></td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td><strong>P_3</strong></td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

- The Gantt chart is:

<table>
<thead>
<tr>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>160</td>
<td>180</td>
</tr>
</tbody>
</table>

- Average waiting time:
- Compare to FCFS and SJF:
Alternating Sequence of CPU And I/O Bursts

```
: : 
load store
add store
read from file

wait for I/O

store increment
index
write to file

wait for I/O

load store
add store
read from file

wait for I/O

: : 
CPU burst

I/O burst

CPU burst

I/O burst

CPU burst

I/O burst
```
Time slice and Context Switch Time

process time = 10

<table>
<thead>
<tr>
<th>quantum</th>
<th>context switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9 10