Resource Allocation Graph describing the traffic jam
Conditions for Deadlock

<table>
<thead>
<tr>
<th>Mutual Exclusion</th>
<th>Hold-and-Wait</th>
<th>No Pre-emption</th>
<th>Circular Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>A process cannot access a resource that has been allocated to another process</td>
<td>a process may hold allocated resources while awaiting assignment of others</td>
<td>no resource can be forcibly removed from a process holding it</td>
<td>a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain</td>
</tr>
</tbody>
</table>
Dealing with Deadlock

• Three general approaches exist for dealing with deadlock:

Prevent Deadlock
  • adopt a policy that eliminates one of the conditions
  • E.g., numbering resources, and request from low to high

Avoid Deadlock
  • make the appropriate dynamic choices based on the current state of resource allocation
  • E.g., banker’s algorithm

Detect Deadlock and Recover
  • attempt to detect the presence of deadlock and take action to recover
Main Points

• Scheduling policy: what to do next, when there are multiple threads ready to run
  – Or multiple packets to send, or web requests to serve, or …
• Definitions
  – response time, throughput
• Uniprocessor policies
  – FIFO
  – Shortest Job First
  – Round robin
  – Multilevel feedback queue
• Multiprocessor policies

Some of the slides are courtesy of Dr. Thomas Anderson
Example

• You manage a restaurant, and the customers complain that they wait forever and starve. What will you do?

• You manage a web site, that suddenly becomes wildly popular. Do you?
  – Buy more hardware?
  – Turn away some users?
  – Implement a different scheduling policy?
Non-preemptive vs Pre-emptive

- A preemptive scheduling means that the scheduler can take resources (e.g., CPU) away from the process.
- A non-preemptive scheduling means a process occupies the resources until it voluntarily relinquishes the resources.
Metrics to evaluate a scheduler

- **Response time**
  - Time elapsed from the time of submission to the first response
- **Throughput**
  - # of tasks can be done per unit of time?
- **Turnaround time**
  - Time elapsed from the time of submission to completion
- **Wait time**
  - Time spent on waiting in the ready queue
- **Predictability (low variance)**
  - How consistent is the performance over time?
- **Fairness**
Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are following:

1. **First Come First Serve (FCFS) Scheduling**
   - Jobs are executed on first come, first serve basis.
   - Easy to understand and implement.
   - Poor in performance as average wait time is high.

2. **Shortest-Job-First (SJF) Scheduling**
   - Best approach to minimize waiting time.
   - Actual time taken by the process is already known to processor.
   - Impossible to implement.

---

Example

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>BURST TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>21</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>2</td>
</tr>
</tbody>
</table>

This is the GANTT chart for the above processes.
First Come First Serve (FCFS)

- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Easy to implement; very small overhead due to scheduling
- The scheduler ignores the property of tasks, so the overall performance, e.g., throughput, is usually poor
Shortest-Process-First (SPF) Scheduling

- Scheduler selects process with smallest time to finish
- Advantages: low average wait time
- Disadvantages:
  - Potentially large variance in wait times
    - Long task can be affected by short tasks once and again
    - Evan, starvation
  - Relies on estimates of time-to-completion
    - Can be inaccurate or unrealistic
Example

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>BURST TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>21</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>2</td>
</tr>
</tbody>
</table>

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following:

```
0 2 5 11 32
P4 P2 P3 P1
```

Now, the average waiting time will be = \( \frac{0 + 2 + 5 + 11}{4} \)/4 = 4.5 ms
Round Robin

• Each task gets resource for a fixed period of time (time quantum)
  – If task doesn’t complete, it goes back in line

• Need to pick a time quantum
  – What if time quantum is too long?
    • Becomes FIFO
  – What if time quantum is too short?
    • Large overhead for context switch

• Advantage: fairness

• Disadvantage: many context switches bring a large overhead; large wait time
Round Robin = Fairness?

• Is Round Robin always fair?
  – No! See the next slides
CPU bound vs I/O bound

• If a process’s speed is mainly determined by the CPU speed, the process is CPU-bound
  – E.g., multiply matrix

• If a process’s speed is mainly determined by the I/O speed (i.e., most of the time the process blocks for I/O), the process is I/O-bound
  – E.g., emacs
Mixed Workload

Tasks

I/O Bound

Issues

I/O Request

I/O Completes

Issues

I/O Request

I/O Completes

CPU Bound

CPU Bound

Time
Multi-level Feedback Queue (MFQ)

- Used in Linux, Windows, MacOS, and Solaris
  - Each system adopts the algorithm with some little modifications
- First developed by Corbato et al; led to Turing Award
MFQ

• Has a set of Round Robin queues
  – Each queue has a separate priority
• High priority queues have short time slices, while low priority queues have long time slices
• Scheduler picks the first thread in highest-priority non-empty queue
• When a process is scheduled out, it is inserted in queues following the two rules
  – If time slice expires, task drops one level
  – If the process relinquishes the slice due to I/O, it is kept in the current priority queue
• Optionally, for a process in the base level queue that becomes I/O bound, it can be promoted to the next-higher queue
<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Slice (ms)</th>
<th>Round Robin Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
Multiprocessor Scheduling

• What would happen if we used MFQ on a multiprocessor?
  – Contention for scheduler spinlock
  – Poor CPU cache reuse
Per-Processor Affinity Scheduling

- Each processor has its own MFQ
  - Protected by a per-processor spinlock
- When the system puts threads back on the ready list, they are put back where they had most recently run
- But idle processors can steal work from other processors
Per-Processor Multi-level Feedback with Affinity Scheduling

Processor 1  
Processor 2  
Processor 3
Summary

• Scheduling policy: what to do next, when there are multiple threads ready to run
• Response time, throughput, wait time
• Uniprocessor policies
  – FIFO, Shortest Job First
  – round robin
  – multilevel feedback as approximation of optimal
• Multiprocessor policies