Today
- Introduction to scheduling
- Classical algorithms

Next Time
- Advanced topics on scheduling
Scheduling out there

- You are the manager of a supermarket (ok, things don’t always turn out how we plan them!)

- It’s a busy time at 5-6PM and you have one register working; how do you optimize the queue to reduce waiting time?
  - You have a handful of customers waiting, each with about equally filled carts
  - A new guy, apparently planning to go on hiding, is now in front of the queue and a bunch of people with 2-3 items wait behind
  - An 8-month expectant mother has joined the back of the queue
  - ...
Scheduling

- Problem
  - Several ready processes and much fewer CPUs
- A choice has to be made
  - By the scheduler, using a scheduling algorithm
- The decision, scheduling, is policy
- Context switching is a mechanism
Scheduling through time

- Early batch systems – Just run the next job in the tape
- Early timesharing systems – Scarce CPU time so scheduling is critical
- PCs – Commonly one active process so scheduling is easy; with fast & per-user CPU scheduling is not critical
- Networked workstations & servers – All back again, multiple ready processes & expensive CS, scheduling is critical
Environments and goals

- Different scheduling algorithms, with different goals, for different application areas
  - Batch
  - Interactive
  - Real-time

- Goals for all/most systems
  - Fairness – comparable processes getting comparable service
  - Policy enforcement – seeing that stated policy is carried out
  - Balance – keeping all parts of the system busy (mix pool of processes)
Environments and goals

- Batch systems
  - CPU utilization – keep CPU busy all the time \((\text{anything wrong?})\)
  - Throughput – max. jobs per hour
  - Turnaround time – min. time bet/ submission & termination

- Interactive systems
  - Response time – respond to requests quickly (time to start responding)
  - Proportionality – meet users’ expectations

- Real-time system
  - Meeting deadlines – avoid losing data
  - Predictability – avoid quality degradation in multimedia systems

- Average, maximum, minimum or \(\text{variance}\)?
Process behavior

- **Task** – a request to be scheduled (a process may be responsible for multiple tasks)
- **Workload** – a set of tasks for some systems to perform, the input to the scheduling algorithm
- **Bursts of CPU usage alternate with periods of I/O wait**
  - Key to scheduling – CPU-bound & I/O bound process
  - As CPU gets faster – more I/O bound processes

![Histogram of CPU-burst times](image)

- Large number of short CPU bursts
- Small number of long CPU bursts

Long CPU burst  Waiting for I/O  Short CPU burst
When to schedule?

- When to make scheduling decisions?
  1. At process creation
  2. When a process exits
  3. When a process blocks on I/O, a semaphore, etc
  4. When an I/O interrupts occurs
  5. A fix periods of time – Need a HW clock interrupting
When to schedule?

- A fixed periods of times … preemptive and non-preemptive
  
  - No-preemptive
    
    • Once a process gets the CPU, it doesn’t release it until the process terminates or switches to waiting
  
  - Preemptive
    
    • Using a timer, the OS can preempt the CPU even if the thread doesn’t relinquish it voluntarily
    
    • Of course, re-assignment involves overhead
And now some example policies

- Remember these are *example policies* – in practice, any real system uses some hybrid approach
- We will start comparing them based on turnaround time

\[
T_{\text{turnaround}} = T_{\text{completion}} - T_{\text{arrival}}
\]

All processes arrived at 0

\[
P_1 \text{ turnaround} = 30 - 0 = 30u
\]
And now some example policies

- Other possible metrics (sometimes in conflict)
  - Maximize *CPU utilization*
  - Maximize *throughput* (requests completed / sec)
  - Minimize *average response time* (avg. time from submission of request to first response)
  - Minimize *energy* (joules per instruction) subject to some constraint (e.g., frames/sec)
First-Come First-Served scheduling

- First-Come First-Served (FCFS)
  - Simplest, easy to implement, non-preemptive

Different burst times

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>

Turnaround time:
(10 + 20 + 30)/3 = 20

Turnaround time:
(30 + 3 + 6)/3 = 13
FCFS Issues

- **The convoy effect**
  - 1 CPU-bound process (burst of 1 sec.)
  - Many I/O-bound ones (needing to read 1000 records)
  - Each I/O-bound process reads one block per sec!

- Potentially bad average response time

- May lead to poor utilization of resources
  - Poor overlap of CPU and I/O

---

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>

Turnaround time:
\[ \frac{(24 + 27 + 30)}{3} = 27 \]
Shortest Job First

- Taken from Operation Research

![Job Schedule Diagram]

Turnaround time:
\[(30 + 3 + 6)/3 = 13\]

- Provably optimal wrt average response time

First job finishes at time \(a\); second job at time \(a + b\); …

Mean turnaround time
\[\frac{(4a + 3b + 2c + d)}{4}\]

Biggest contributor

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Job #</th>
<th>Finish time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
</tbody>
</table>
And now a short break …
Shortest Job First

- Another example

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

Turnaround time: \((16 + 5 + 1 + 9)/4 = 7.75\)

- What if they don’t all arrive at the same time?

Turnaround time: \((7 + 11 + 6 + 13)/4 = 9.25\)

*Note P2 run at 12 but arrived at 1, so it only waited 11; similar with P3 and P4.*
Shortest Remaining Time First

- A preemptive variation

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>P2</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>2.0</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>3.0</td>
<td>4</td>
</tr>
</tbody>
</table>

Turnaround time: \( \frac{(16 + 5 + 1 + 7)}{4} = 7.25 \)

- Great, but how do you know the burst time?
Determining length of next CPU burst

- Can only *estimate* length
- Typically done using length of previous CPU bursts and exponential averaging

\[- t_n = \text{actual length of } n^{th} \text{ CPU burst}\]
\[- \tau_{n+1} = \text{predicted value for the next CPU burst}\]
\[- \alpha, \; 0 \leq \alpha \leq 1\]

- Define:

\[
\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n.
\]

\[
\tau_1 = 0.5 \cdot t_0 + (1-0.5)\tau_0 = 8
\]
Scheduling the server-side of P2P systems

- P2P users’ response is dominated by download
  - >80% download requests in Kazaa are rejected due to capacity saturation at server peers
  - >50% of all requests for large objects (>100MB) take more than one day & ~20% take over one week to complete

- Most implementations use FCFS or PS
  - Apply SRPT!

Work from Northwestern

Mean response time of object download as a function of system load.

Legend:
- PS – Process Sharing
- FCFS – First-Come First-Serve
- SRPT – Shortest-Remaining Processing-Time
Priority scheduling

- SJF is a special case of priority-based scheduling
  - Priority = reverse of predicted next CPU burst
- Pick process with highest priority (lowest number)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Turnaround time = \(\frac{(16 + 1 + 18 + 19 + 6)}{5} = 12\)
Priority scheduling issues

- How do you assign priorities?
- Starvation
  - With an endless supply of high priority jobs, low priority processes may never execute
  - *What other recently discussed algorithm has the same problem?*
- Solution
  - Increases priority with age, i.e. accumulated waiting
  - Decrease priority as a function of accumulated processing time
  - Assigned maximum quantum
Round-robin scheduling

- SJF is not bad if you know burst times or can estimate it fairly well – the case in many early batch systems
  - At least when measuring turnaround time!
- Time-sharing machines changed it all
  - Users want interactivity
  - Turnaround time is not a good metric for this
  - *Response time?* Time to first run minus time of arrival

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

Turnaround time: \((16 + 5 + 1 + 9)/4\) \(=\) 7.75

Response time: \((9 + 1 + 0 + 5)/4\) \(=\) 3.75
Round-robin scheduling

- Simple, fair, easy to implement, & widely-used
- Each process gets a fix quantum or time slice
- When quantum expires, if running preempt CPU
- With $n$ processes & quantum $q$, each one gets $1/n$ of the CPU time, no-one waits more than $(n-1)q$ to run first (i.e., response time)

$q = 2$

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

Response time: \((0 + 2 + 4 + 5)/4 = 2.75\)

Turnaround time: \((16 + 11 + 5 + 13)/4 = 11.25\)
Quantum & Turnaround time

- **Length of quantum**
  - Too short – low CPU efficiency (*why?*)
  - Too long – low response time (*really long, what do you get?*)
  - Commonly ~ 50-100 msec.

![Graph of average turnaround time vs. time for processes P1 to P4]
Next time

- How do you support responsive, flexible scheduling? Priority? How are priorities set?
- How do you optimize turnaround time while minimizing response time?
  - Shortest Job First reduces turnaround time but hurts response time
  - Round Robin reduces response time but hurts average waiting time
  - …?