Scheduling I

To do ...

- Introduction to scheduling
- Classical algorithms
- Next Time: Advanced topics on scheduling
You are the manager of a supermarket (ok, things don’t always turn out the way we plan them!)

It’s a busy time at 5-6PM and you have one register working; how do you manage the queue to reduce waiting time?

– You have a handful of customers waiting, each with about equally filled carts
– A guy, apparently planning to go into 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, scheduling is critical
- PCs – Commonly one active process, scheduling is easy; with fast and per-user CPU, scheduling is not critical
- Networked workstations, servers, data centers – All back again, multiple ready processes and expensive context switching, scheduling is critical
- Mobile devices, battery life overhead so can the scheduler help
- …
Environments and goals

- Different scheduling algorithms, with different goals, for different application areas
  - Batch, interactive, real-time

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

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

- …
Environments and goals

... 

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

- **Average, maximum, minimum or variance?**
  - Throughput, turnaround time, response 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)
Process behavior

- **Task** – a request to scheduled (multiple tasks/process)
- **Workload** – set of tasks to run, input to sched 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)

- Long CPU burst
- Waiting for I/O
- Short CPU burst

- Large number of short CPU bursts
- Small number of long CPU bursts
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 or not
  - No-preemptive
    - Once a process gets the CPU, it doesn’t release it until the process terminates or switches to waiting
    - It may take hours to finish, that’s OK
  - Preemptive
    - Let a process run for a maximum fixed time, if running still the OS can preempt the CPU
    - This requires having a clock interrupt at the end of the interval
Comparing policies

- We’ll see some example policies – in practice, any real system uses some hybrid approach
- For comparison – a metric: turnaround time (performance)

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>20</td>
</tr>
<tr>
<td>P2</td>
<td>204</td>
</tr>
<tr>
<td>P3</td>
<td>8</td>
</tr>
</tbody>
</table>

P1’s turnaround = 30 – 0 = 30u
Comparing policies

- Other metrics/goals (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)
And now a short break ...
First-Come First-Served scheduling

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>10</td>
</tr>
<tr>
<td>P3</td>
<td>10</td>
</tr>
</tbody>
</table>

With 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>

Avg turnaround time: 
\[(10 + 20 + 30)/3 = 20\]
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

**Remember the guy going into hiding?**

<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 (SJF)

- Taken from Operation Research

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

- Provably optimal wrt turnaround time

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

Mean turnaround time
\[(4a + 3b + 2c + d)/4\]
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?

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

*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: \((16 + 5 + 1 + 7)/4 = 7.25\)

- 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:

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

\[ \tau_1 = 0.5 t_0 + (1 - 0.5)\tau_0 = 8 \]
Priority scheduling

- SJF, a special case of priority-based sched
  - Priority = reverse of predicted next CPU burst
- Pick process with highest priority (lowest #)

<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?

• Possible starvation
  – With an endless supply of high priority jobs, low priority processes may never execute
  – *What other recently discussed algorithm has the same problem?*

• Solutions
  • Increases priority with age, i.e. accumulated waiting
  • Lower priority as a function of acc’ed 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

Turnaround time: \(\frac{16 + 5 + 1 + 9}{4} = 7.75\)

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

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

$$q = 2$$

Response time: $\frac{0 + 2 + 4 + 5}{4} = 2.75$

Turnaround time: $\frac{16 + 11 + 5 + 13}{4} = 11.25$
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.

<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>6</td>
</tr>
<tr>
<td>P₂</td>
<td>3</td>
</tr>
<tr>
<td>P₃</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>7</td>
</tr>
</tbody>
</table>

Larger quantum doesn’t imply shorter turnaround times
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
- …?