Lecture 7:

Processes II: CPU Scheduling

www.cl.cam.ac.uk/Teaching/2001/OSFounds/

Today’s Lecture

Today we’ll cover:
- How do we schedule the CPU?
  - Criteria, and
  - Various strategies.

CPU-I/O Burst Cycle

- **CPU-I/O Burst Cycle**: process execution consists of a **cycle** of CPU execution and I/O wait.

- Processes can be described as either:
  1. **I/O-bound**: spends more time doing I/O than computation; has many short CPU bursts.
  2. **CPU-bound**: spends more time doing computations; has few very long CPU bursts.

- Observe most processes execute for at most a few milliseconds before blocking
  ⇒ need **multiprogramming** to obtain decent overall CPU utilization.

CPU Scheduler

Recall: CPU scheduler selects one of the **ready** processes and allocates the CPU to it.
- There are a number of occasions when we can/must choose a new process to run:
  1. a running process blocks (running → blocked)
  2. a timer expires (running → ready)
  3. a waiting process unblocks (blocked → ready)
  4. a process terminates (running → exit)

- If only make scheduling decision under 1, 4 ⇒ have a **non-preemptive** scheduler:
  - simple to implement
  - open to denial of service
    - e.g. Windows 3.11, early MacOS.

- Otherwise the scheduler is **preemptive**.
  - solves denial of service problem
  - more complicated to implement
  - introduces concurrency problems...
**Idle system**

What do we do if there is no ready process?

- **halt processor** (until interrupt arrives)
- saves power (and heat!)
- increases processor lifetime
  - might take too long to stop and start.
- **busy wait** in scheduler
- quick response time
- ugly, useless
- **invent idle process**, always available to run
- gives uniform structure
- could use it to run checks
- uses some memory
- can slow interrupt response

In general there is a trade-off between responsiveness and usefulness.

**Scheduling Criteria**

A variety of metrics may be used:

1. **CPU utilization**: the fraction of the time the CPU is being used (and not for idle process!)
2. **Throughput**: # of processes that complete their execution per time unit,
3. **Turnaround time**: amount of time to execute a particular process.
4. **Waiting time**: amount of time a process has been waiting in the ready queue.
5. **Response time**: amount of time it takes from when a request was submitted until the first response is produced (in time-sharing systems)

Sensible scheduling strategies might be:

- Maximize throughput or CPU utilization
- Minimize average turnaround time, waiting time or response time.

Also need to worry about **fairness** and **liveness**.

**First-Come First-Served Scheduling**

- FCFS depends on order processes arrive, e.g.

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>25</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>7</td>
</tr>
</tbody>
</table>

- If processes arrive in the order $P_1$, $P_2$, $P_3$:

  - Waiting time for $P_1=0$; $P_2=25$; $P_3=29$;
  - Average waiting time: $(0 + 25 + 29)/3 = 18$.

- If processes arrive in the order $P_3$, $P_2$, $P_1$:

  - Waiting time for $P_1=11$; $P_2=7$; $P_3=0$;
  - Average waiting time: $(11 + 7 + 0)/3 = 6$.
  - i.e. three times as good!

- First case poor due to **convoy effect**.

**SJF Scheduling**

Intuition from FCFS leads us to **shortest job first** (SJF) scheduling.

- Associate with each process the length of its next CPU burst.
- Use these lengths to schedule the process with the **shortest time** (FCFS can be used to break ties).

For example:

<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>

- Waiting time for $P_1=0$; $P_2=6$; $P_3=3$; $P_4=7$;
- Average waiting time: $(0 + 6 + 3 + 7)/4 = 4$.

SJF is optimal in that it gives the minimum average waiting time for a given set of processes.
SRTF Scheduling

- SRTF = Shortest Remaining-Time First.
- Just a preemptive version of SJF.
- i.e. if a new process arrives with a CPU burst length less than the remaining time of the current executing process, preempt.

For example:

<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></td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 9$; $P_2 = 1$; $P_3 = 0$; $P_4 = 2$;
- Average waiting time: $(9 + 1 + 0 + 2)/4 = 3$.

What are the problems here?

Predicting Burst Lengths

- For both SJF and SRTF require the next "burst length" for each process = need to estimate it.
- Can be done by using the length of previous CPU bursts, using exponential averaging:
  1. $t_n$ = actual length of $n^{th}$ CPU burst.
  2. $\tau_{n+1}$ = predicted value for next CPU burst.
  3. For $\alpha, 0 \leq \alpha \leq 1$ define:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

- If we expand the formula we get:

$$\tau_{n+1} = \alpha t_n + \ldots + (1 - \alpha)^2 \alpha t_{n-j} + \ldots + (1 - \alpha)^n \tau_0$$

where $\tau_0$ is some constant.

- Choose value of $\alpha$ according to our belief about the system, e.g. if we believe history irrelevant, choose $\alpha = 1$ and then get $\tau_{n+1} = t_n$.

- In general an exponential averaging scheme is a good predictor if the variance is small.

Round Robin Scheduling

Define a small fixed unit of time called a quantum (or time-slice), typically 10-100 milliseconds. Then:

- Process at the front of the ready queue is allocated the CPU for (up to) one quantum.
- When the time has elapsed, the process is preempted and appended to the ready queue.

Round robin has some nice properties:

- Fair: if there are $n$ processes in the ready queue and the time quantum is $q$, then each process gets $1/n^{th}$ of the CPU.
- Live: no process waits more than $(n - 1)q$ time units before receiving a CPU allocation.
- Typically get higher average turnaround time than SRTF, but better average response time.

But tricky choosing correct size quantum:

- $q$ too large $\Rightarrow$ FCFS/FIFO
- $q$ too small $\Rightarrow$ context switch overhead too high.

Static Priority Scheduling

- Associate an (integer) priority with each process
- For example:

<table>
<thead>
<tr>
<th>Priority</th>
<th>Process Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>system internal processes</td>
</tr>
<tr>
<td>1</td>
<td>interactive processes (staff)</td>
</tr>
<tr>
<td>2</td>
<td>interactive processes (students)</td>
</tr>
<tr>
<td>3</td>
<td>batch processes.</td>
</tr>
</tbody>
</table>

- Then allocate CPU to the highest priority process:
  - 'highest priority' typically means smallest integer
  - get preemptive and non-preemptive variants.
- e.g. SJF is a priority scheduling algorithm where priority is the predicted next CPU burst time.

- Problem: how to resolve ties?
  - round robin with time-slicing
  - allocate quantum to each process in turn.
  - Problem: biased towards CPU intensive jobs.
    * per-process quantum based on usage?
    * ignore?
- Problem: starvation...
**Dynamic Priority Scheduling**

Use same scheduling algorithm, but allow priorities to change over time, e.g.

1. **Simple aging:**
   - processes have a (static) base priority and a dynamic effective priority.
   - if process starved for $k$ seconds, increment effective priority.
   - once process runs, reset effective priority.

2. **Computed priority:**
   - first used in Dijkstra's THE
   - time slots: $t$, $t + 1$, ...
   - in each time slot $t$, measure the CPU usage of process $j$: $u^t$
   - priority for process $j$ in slot $t + 1$:
     \[
     p^t_{j+1} = f(u^t_j, p^t_j, u^t_{j-1}, p^t_{j-1}, \ldots)
     \]
   - e.g. $p^t_{j+1} = p^t_j/2 + ku^t_j$
   - penalises CPU bound $\rightarrow$ supports I/O bound.

Today such computation considered acceptable. . .

---

**Summary**

You should now understand:

- What a CPU scheduler does.
- Criteria for scheduling.
- Predicting burst lengths.
- Various strategies:
  1. First-come first-served,
  2. Shortest job first,
  3. Shortest remaining-time first,
  4. Round-robin,
  5. Static and dynamic priorities.

Next lecture: Memory Management

**Background Reading:**

- Silberschatz et al.: – Chapter 6