From last time

Does each of the following appear in processes, programs, both, or neither?
– instructions
– read-only data
– registers
– a stack
– a heap
– network connections
– system calls
– a shared data area

COMP25111: Operating Systems
Lecture 6: An Introduction to Process (and Thread) Scheduling

Will Toms
School of Computer Science, University of Manchester
Autumn 2016

Overview & Learning Outcomes

Scheduling
– When to change process
– How to select the next process to run
– Criteria

Process States & Transitions

Basic Concepts

Scheduler:
– component of OS process manager
– decides which “ready” process to run next
 1 per CPU (core)
– “scheduling algorithm”
– processes or kernel-level threads
– CPU time was expensive, so scheduling very important
– PC (1 user & cheap) but scheduling ever more sophisticated

When to schedule?

When a process frees the CPU
When a new process joins the “ready” list

CPU burst: executing on CPU
I/O burst: blocked, waiting for I/O

Process alternates between CPU & I/O bursts

CPU bound: long CPU bursts
I/O bound: short CPU bursts

A very long CPU burst keeps other processes waiting
Example

Two processes, A and B, arrive at time 0;
CPU-burst time length: A=10, B=4 time-units (total=14)
e.g. run A then B
A: running
B: ready

Average turnaround time: A,B (10+14)/2=12; B,A (14+4)/2=9;
Average waiting time: A,B (0+10)/2=5; B,A (4+0)/2=2;

First-Come-First-Served (FCFS)

Simplest CPU scheduling algorithm
First process in ready state gets CPU first & runs until blocked (or finished).

Requires a single queue of ready processes:
– add ready process to queue tail
– if the CPU is free, run process at queue head
e.g. A is CPU-bound
B & C are I/O-bound
A: running I/O ready running I/O ready running...
B: ready run I/O ready run I/O ready...
C: ready run I/O ready run I/O ready...

(“I/O” = blocked, waiting for I/O to complete)
B and C spend too much time “ready”

Preemptive vs Non-Preemptive Scheduling

Non-preemptive scheduling: process runs until terminated or “blocked”
To avoid a process with a very long CPU-burst hogging CPU:

Preemptive scheduling: process can run (continuously) for some fixed maximum time;
if it reaches the maximum time, it is interrupted & set “ready” and the scheduler runs another process
needs timer interrupt
fixed amount of time = “time-slice” or “time quantum”
e.g. 10-100msec

Appropriate length for time quantum?

Question - FCFS scheduling

Process A arrives at time 0: 4 time-units CPU, then 2 I/O, then 3 CPU, then 2 I/O
Process B arrives at time 1: 3 time-units CPU, then 1 I/O, then 1 CPU, then 1 I/O

A preemptive version of the first example

Two processes, A and B, arrive at time 0;
CPU-burst time length: A=10, B=4 time-units
time-slice = 5 time-units

3rd case: A then B, & time-slice=5
A: ready running
B: running

Average turnaround time: 3rd: (14+9)/2=11.5 (was 12 or 9)
Average waiting time: 3rd: (4+5)/2=4.5 (was 5 or 2)

Round-Robin Scheduling

Simplest algorithm for time-sharing systems
Improves average turnaround time & waiting time
Question - Round-Robin scheduling

Process A arrives at time 0: 4 time-units CPU, then 2 I/O, then 3 CPU, then 2 I/O
Process B arrives at time 1: 3 time-units CPU, then 1 I/O, then 1 CPU, then 1 I/O

time-slice = 1 time-unit

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Length of time-slice?

Two identical processes: 3 time-units CPU, then 4 time-units I/O, then 3 time-units CPU

Time-slice = 3:

<table>
<thead>
<tr>
<th></th>
<th>run</th>
<th>blocked</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>3 4</td>
<td>6 6 7 8</td>
<td>9 10 11 12 13 14</td>
</tr>
</tbody>
</table>

Question: Time-slice = 2:

<table>
<thead>
<tr>
<th></th>
<th>run</th>
<th>blocked</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>3 4</td>
<td>6 6 7 8</td>
<td>9 10 11 12 13 14</td>
</tr>
</tbody>
</table>

Scheduling Goals / Criteria for Evaluation

General, chose scheduling algorithms with most typical behaviour that best satisfies most desirable criteria:

- Fairness
- high CPU utilisation
- high Throughput
- low Turnaround time
- low Waiting time
- low Response time
- Meeting deadlines
- Prioritisation
- etc.

What is a good choice for the time quantum?

Different time slices may lead to different results (e.g. above)
The major trade-off is the cost of a context switch – ignored so far.
(i.e. save process state, pick new process, restore its state)
If the time slice is smaller, the cost of the context switch (CS) becomes more significant e.g.:

<table>
<thead>
<tr>
<th></th>
<th>running</th>
<th>CS</th>
<th>ready</th>
<th>CS</th>
<th>running</th>
<th>CS</th>
<th>ready...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ready</td>
<td>CS</td>
<td>running</td>
<td>CS</td>
<td>ready</td>
<td>CS</td>
<td>running...</td>
</tr>
</tbody>
</table>

time-slice:context-switch = 3:1 → 25% CPU time lost

Two solutions:
- increase the time slice (but too much will make Round-Robin look like FCFS);
- reduce the cost of context switch (H/W support?)
e.g. quantum = 20-50ms, context switch < 1ms

Non-preemptive – Shortest-Job-First

In the first example, starting with shortest job minimised average turnaround & waiting time.

Generalise: given a set of ready processes, run the one that has the smallest CPU burst
e.g. processes A, B, C, D arrive at time 0,
CPU bursts: A=7, B=4, C=9, D=5

<table>
<thead>
<tr>
<th></th>
<th>ready</th>
<th>run 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>run 4</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>ready</td>
<td>run 9</td>
</tr>
<tr>
<td>D</td>
<td>ready</td>
<td>run 5</td>
</tr>
</tbody>
</table>

Summary of key points

Scheduler: chooses a process from the ready state to run

Preemptive & Non-Preemptive

Algorithms: FCFS, Round-Robin, Shortest-First
- typical behaviour?

Criteria: fairness, utilisation, throughput, turnaround, wait, response, ...
- most desirable in given situation?

Next lecture: more process scheduling – (priority-based)
Your Questions

For next time

Exam Questions

Glossary

Reading