Chapter 5
Scheduling

Images from Silberschatz
CPU usage/IO bursts

- Life time of a single process

- What would an IO bound process look like?

- What would a CPU bound process look like?
- Single process
- What would an IO bound process look like?
- What would a CPU bound process look like?
CPU Scheduler

- Short term scheduler

- Takes process from ready queue and runs it
  - Various algorithms used here.....
  - Data structure? Why?
  - Puts it on the CPU

- Takes a process off the CPU and puts it on the ready queue
  - Why?

- Swapping processes around causes a ......
Scheduling events

• Processes moved from the CPU when:
  – Switches from running to waiting state
  – Switches from running to ready state
  – Switches from waiting to ready
  – Terminates

  – What if only first and last are implemented?
    • Why would we ever do this?
Problems

- What happens if a process is preempted while in a system call?
  - Possible bad outcomes?

- How to fix this?
Dispatcher

- Module/code that puts the process on the CPU
  - Switch context
  - Switch to user mode
  - Restart at correct program counter (PC)

- Dispatch Latency:
Goals

- CPU Utilization
- Throughput
- Turnaround time
- Waiting time
- Response time

- Usually optimize average
  - Sometimes optimize the minimum or maximum
  - Sometimes minimize the variance
  - Why? Which values?
Scheduling Algorithms

- First-Come, First-Served (FCFS)
  - Non-preemptive (cooperative!)
  - Data structure?

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

Suppose that the processes arrive in the order: $P_1$, $P_2$, $P_3$. The Gantt Chart for the schedule is:

- Waiting time
- Average waiting time:
FCFS, cont

Suppose that the processes arrive in the order $P_2, P_3, P_1$

- The Gantt chart for the schedule is:

![Gantt chart]

- Waiting time
- Average waiting time:

- *Convoy effect*

- Advantages?
Shortest Job First (SJR)

- Choose process who's next CPU burst is the shortest
  - Not really shortest JOB first

- May be preemptive (or not)
  - Preemptive (Shortest-Remaining-Time-First (SRTF))

- Gives minimum average waiting time
  - Provably optimal
  - Preemptive
  - With perfect knowledge
Example (cooperative!)

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive)

- Average waiting time
Preemptive

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (preemptive)

- Average waiting time
Why is this hard?

- Length of next CPU burst is?

1. $t_n = \text{actual length of } n^{th} \text{ CPU burst}$

2. $\tau_{n+1} = \text{predicted value for the next CPU burst}$

3. $\alpha, 0 \leq \alpha \leq 1$

4. Define: $\tau_{n=1} = \alpha t_n + (1 - \alpha) \tau_n$.

- Why? What does this mean? What does this look like?
Prediction of next CPU Burst

CPU burst \( (t_i) \) \hspace{1cm} 6 \hspace{0.5cm} 4 \hspace{0.5cm} 6 \hspace{0.5cm} 4 \hspace{0.5cm} 13 \hspace{0.5cm} 13 \hspace{0.5cm} 13 \hspace{0.5cm} \ldots

"guess" \( (\tau_i) \) \hspace{1cm} 10 \hspace{0.5cm} 8 \hspace{0.5cm} 6 \hspace{0.5cm} 6 \hspace{0.5cm} 5 \hspace{0.5cm} 9 \hspace{0.5cm} 11 \hspace{0.5cm} 12 \hspace{0.5cm} \ldots
Priority Scheduling

- Give each process a priority (an integer)
- Schedule process with highest priority
  - May be the lowest integer (to make things more confusing)

- Preemptive or not

- SJF is a special case of this
  - What is the priority?

- Where might a problem arise?
Round Robin

• Each process gets some amount of time (10-100 milliseconds)
  - Time quantum/slice
  - Put at the end of the queue when done

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>17</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

The Gantt chart is:

```
 0 2 0 3 7 5 7 7 7 9 7 1 1 7 1 2 1 3 4 1 5 4 1 6 2
```

Typically, higher average turnaround than SJF, but better response
Time quanta & context switches

process time = 10

quantum
12
6
1

context switches
0
1
9
Turnaround time

<table>
<thead>
<tr>
<th>process</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>7</td>
</tr>
</tbody>
</table>

The graph shows the average turnaround time for different time quantum settings. The table lists the processes and their corresponding turnaround times.
Multilevel Queue Scheduling

- Different Queues, different algorithms
  - Process stays in one queue forever

- Foreground

- Background

- Other categories
highest priority

- system processes
- interactive processes
- interactive editing processes
- batch processes
- student processes

lowest priority

- Absolute vs Time slicing
Multilevel Feedback-Queue Scheduling

• Processes move between queues
  – Use CPU burst information to move processes
  – Aging may play a role

• Defining characteristics
  – number of queues
  – scheduling algorithms for each queue
  – method used to determine when to upgrade a process
  – method used to determine when to demote a process
  – method used to determine which queue a process will enter when that process needs service
Example (p168)

- Three queues
  - Q0 RR with time quantum 8 milliseconds
  - Q1 RR with time quantum 16 milliseconds
  - Q2 FCFS
Multiple-Processor Scheduling

- Asymmetric Multiprocessor

- Symmetric Multiprocessor

- Processor Affinity
  - Soft vs hard
Cont.

• Load Balancing
  – Push migration

  – Pull migration

• Hyperthreading
Thread Scheduling

- Process-contention-scope

- System-contention-scope
#include <pthread.h>
#include <stdio.h>
#define NUM THREADS 5

int main(int argc, char *argv[]) {
    int i;
    pthread t tid[NUM THREADS];
    pthread attr t attr;
    /* get the default attributes */
    pthread attr init(&attr);
    /* set the scheduling algorithm to PROCESS or SYSTEM */
    pthread attr setscope(&attr, PTHREAD SCOPE SYSTEM);
    /* set the scheduling policy - FIFO, RR, or OTHER */
    pthread attr setschedpolicy(&attr, SCHED OTHER);
    /* create the threads */
    for (i = 0; i < NUM THREADS; i++)
        pthread create(&tid[i],&attr,runner,NULL);
    /* now join on each thread */
    for (i = 0; i < NUM THREADS; i++)
        pthread join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param) {
    printf("I am a thread\n");
    pthread exit(0);
}
Solaris Scheduling

- Priority based
- Classes
  - Real time
  - System
  - Time Sharing
  - Interactive
- Solaris 9 adds
  - Fixed priority
  - Fair share
<table>
<thead>
<tr>
<th>priority</th>
<th>time quantum</th>
<th>time quantum expired</th>
<th>return from sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>160</td>
<td>5</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>25</td>
<td>120</td>
<td>15</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>20</td>
<td>53</td>
</tr>
<tr>
<td>35</td>
<td>80</td>
<td>25</td>
<td>54</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>30</td>
<td>55</td>
</tr>
<tr>
<td>45</td>
<td>40</td>
<td>35</td>
<td>56</td>
</tr>
<tr>
<td>50</td>
<td>40</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>55</td>
<td>40</td>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>20</td>
<td>49</td>
<td>59</td>
</tr>
</tbody>
</table>
Linux

• Preemptive, priority based

• Two priority ranges (lower is better):
  - Real-time: 0-99
  - Nice: 100-140

<table>
<thead>
<tr>
<th>numeric priority</th>
<th>relative priority</th>
<th>time quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>highest</td>
<td>200 ms</td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>lowest</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

active array

priority [0] task lists
[1]        
[

expired array

priority [0] task lists
[1]        
[140]
Algorithm Evaluation

• How to choose a scheduling algorithm?
  - Define goals
    • Minimize wait time? Minimize response time? Maximize CPU utilization?

• Deterministic modeling

• Queuing modeling (queueing network analysis)
  - Little's formula

• Simulations

• Build it