Operating Systems

Process Scheduling
(Ch 3.2, 5.1-5.3)

Schedulers

• Short-Term
  – “Which process gets the CPU?”
  – Fast, since once per 100 ms
• Long-Term (batch)
  – “Which process gets the Ready Queue?”
• Medium-Term
  – “Which Ready Queue process to memory?”
  – Swapping
CPU-IO Burst Cycle

- add
- read
- store
- increment
- write
- (I/O Wait)

Preemptive Scheduling

- Four times to re-schedule
  1. Running to Waiting (I/O wait)
  2. Running to Ready (time slice)
  3. Waiting to Ready (I/O completion)
  4. Termination
- #2 optional ==> “Preemptive”
- Timing may cause unexpected results
  - updating shared variable
  - kernel saving state
Question

• What Criteria Should the Scheduler Use?
  – Ex: favor processes that are small
  – Others?

Scheduling Criteria

• Internal
  – open files
  – memory requirements
  – CPU time used - time slice expired (RR)
  – process age - I/O wait completed

• External
  – $
  – department sponsoring work
  – process importance
  – super-user (root) - nice
Scheduling Measures of Performance

1. CPU utilization (40 to 90)
2. Throughput (processes / hour)
3. Turn-around time
4. Waiting time (in queue)
   • Maximize #1, #2      Minimize #3, #4
   • Response time
     – Self-regulated by users (go home)
     – Bounded ==> Variance!

First-Come, First-Served

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

Gantt Chart

• Avg Wait Time \( \frac{0 + 8 + 9}{3} = 5.7 \)
Shortest Job First

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

- Avg Wait Time \((0 + 1 + 2) / 3 = 1\)
- Optimal Avg Wait
- Prediction tough … Ideas?

Priority Scheduling

- SJF is a special case

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Avg Wait Time \((0 + 1 + 9) / 3 = 3.3\)
Round Robin

• Fixed time-slice and Preemption

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>

• Avg Turnaround = (8 + 9 + 11) / 3 = 9.3
• FCFS? SJF?

SOS: Dispatcher

• What kind of scheduling algorithm is it?
• There is no “return” from the Dispatcher()
  … why?
  – OS system stack
• Why is there a while(1);?
  – Is this infinite loop ok? Why?
Round Robin Fun

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
</tbody>
</table>

- Turn-around time?
  - q = 10
  - q = 1
  - q --> 0

More Round Robin Fun

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
</tr>
</tbody>
</table>

Rule: 80% within one quantum
## Fun with Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Gantt Charts:**
  - FCFS
  - SJF
  - Priority
  - RR (q=1)

- **Performance:**
  - Throughput
  - Waiting time
  - Turnaround time

## More Fun with Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>0.4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Turn around time:**
  - FCFS
  - SJF
  - q=1 CPU idle
  - q=0.5 CPU idle
Multi-Level Queues

• Categories of processes

| Priority 1 | System |
| Priority 2 | Interactive |
| Priority 3 | Batch |

• Run all in 1 first, then 2 …
• Starvation!
• Divide between queues: 70% 1, 20% 2 …

Multi-Level Feedback Queues

• Time slice expensive but want interactive

| Priority 1 | Queue 1 Quantum |
| Priority 2 | Queue 2 Quanta |
| Priority 3 | Queue 4 Quanta |

• Consider process needing 100 quanta
  – 1, 4, 8, 16, 32, 64 = 7 swaps!
• Favor interactive users
Outline

• Processes
  – PCB
  – Interrupt Handlers
• Scheduling
  – Algorithms
  – Linux
  – WinNT/2000

Linux Process Scheduling

• Two classes of processes:
  – Real-Time
  – Normal
• Real-Time:
  – Always run Real-Time above Normal
  – Round-Robin or FIFO
  – “Soft” not “Hard”
Linux Process Scheduling

• Normal: *Credit-Based* (counter variable)
  – process with most credits is selected
    + goodness() function
  – Timer goes off (jiffy, 1 per 10 ms)
    + then lose a credit (0, then suspend)
  – no runnable process (all suspended), add to
    every process:
    - recalculate:
      credits = credits/2 + priority
  • Automatically favors I/O bound processes

Windows Scheduling

• Basic scheduling unit is a thread
  – (Can think if threads as processes for now)
• Priority based scheduling per thread
• Preemptive operating system
• No shortest job first, no quotas
Priority Assignment

- Windows kernel uses 31 priority levels
  - 31 is the highest; 0 is system idle thread
  - Realtime priorities: 16 - 31
  - Dynamic priorities: 1 - 15
- Users specify a priority class:
  + realtime (24), high (13), normal (8) and idle (4)
  - and a relative priority:
    + highest (+2), above normal (+1), normal (0), below normal (-1), and lowest (-2)
  - to establish the starting priority
- Threads also have a current priority

Quantum

- Determines how long a Thread runs once selected
- Varies based on:
  - Workstation or Server
  - Intel or Alpha hardware
  - Foreground/Background application threads (3x)

- How do you think it varies with each?
Dispatcher Ready List

- Keeps track of all Ready-to-execute threads
- Queue of threads assigned to each level

FindReadyThread

- Locates the highest priority thread that is ready to execute
- Scans dispatcher ready list
- Picks front thread in highest priority nonempty queue

- When is this like round robin?
Boosting and Decay

• Boost priority
  – Event that “wakes” blocked thread
  + Amount of boost depends upon what blocked for
    – Ex: keyboard larger boost than disk
  – Boosts never exceed priority 15 for dynamic
    – Realtime priorities are not boosted

• Decay priority
  – by one for each quantum
  – decays only to starting priority (no lower)

Starvation Prevention

• Low priority threads may never execute
• “Anti-CPU starvation policy”
  – thread that has not executed for 3 seconds
  – boost priority to 15
  – double quantum
• Decay is swift not gradual after this boost