Scheduling
Dickinson College
Computer Science 354
Spring 2010

Road Map
- Past:
  - What an OS is, why we have them, what they do.
  - Base hardware and support for operating systems
  - Process Management
  - Threads
- Present:
  - Scheduling
- Future:
  - Synchronization / Concurrent programming
  - Memory management
  - Storage management

Scheduling
- Outline
  - Types of scheduling
  - Context switching
  - Performance metrics
  - CPU Scheduling algorithms
  - Real OS Examples

Types of Scheduling
- There are three types of scheduling that occur on different time scales:
  - CPU scheduling
    - a.k.a. short-term scheduling or just scheduling
  - Job scheduling
    - a.k.a. Long-term scheduling
  - Medium-term scheduling
    - Not every OS uses all three types.

CPU Scheduling
- The short-term CPU scheduler (or just scheduler) selects, from the ready queue, the next process/thread that will run on the CPU.
- The CPU scheduler may run 10-100 times per second.

Job Scheduling
- A job scheduler selects jobs (i.e., processes) from a pool of new, but as yet unexecuted, processes to add to the ready queue.
- Job scheduling may only occur every several minutes.
Job Scheduling Criterion

- The job scheduler determines the degree of multiprogramming in the system.
- The job scheduler attempts to maintain a balance of CPU Bound processes and I/O Bound processes in the system.

Medium-term Scheduling

- The medium-term scheduler swaps partially executed processes in and out of RAM to alter the process mix and adjust the degree of multiprogramming.

Random OS Cartoon

Context Switching

- Any time the CPU is switched from running one process to running another process, a context switch must occur.
  - A context switch is a two part process:
    - The context of the running process must be saved.
    - The context of the next process to run must be restored.

Saving a Process’ Context

- A running process’ context is saved immediately following any interrupt or system call.
  - Saving the context requires backing up any values that may/will be overwritten by another process.
  - Values are backed up by copying them to the process’ PCB or the thread’s TCB.
  - Some of the values that are copied include:
    - The value of the PC just before the interrupt or system call
    - The current values in the SP and general purpose registers.
    - Address space information (base/limit registers or VM information)

Restoring a Process’ Context

- Restoring a context:
  - A process’ context is restored just before control of the CPU is given to that process.
    - The values of the SP and general purpose registers are copied from the PCB or TCB back into the registers.
    - The machine is switched to user mode.
    - The PC is set to the value from the PCB or TCB.
Speeding up Context Switches

- Context switching takes time.
- Some techniques that have been used to speedup context switching include:
  - Partial context switches
  - Kernel register set
  - Multiple register sets
  - Machine language instruction
  - Threads

CPU Scheduling

- In designing the CPU scheduler there are two major design questions that must be answered:
  - Under what circumstances will the scheduler be invoked?
    - Non-preemptive vs. Preemptive scheduling
  - When the scheduler is invoked, what criterion will it use to select, from the ready queue, the next process to run?
    - Scheduling Algorithms

Scheduling Opportunities

- There are four opportunities for the CPU scheduler to select a new process to run:
  1. The running process blocks. (running → waiting)
  2. A new process is created. (new → ready)
    Or started by the long term scheduler.
  3. The running process is interrupted. (running → ready)
    Or yields.
    A process may also unblock. (waiting → ready)
  4. A process exits. (running → terminated)

Non-Preemptive vs. Preemptive

- Depending upon which scheduling opportunities are used by a scheduler the scheduling can be:
  - Non-Preemptive: The scheduler will allow the running process to continue to run as long as it remains ready (i.e. doesn’t block or exit).
  - Preemptive: The scheduler may set aside the running process in favor or another at any scheduling opportunity.

Preemptive Scheduling

- Preemptive scheduling enables a number of things that non-preemptive scheduling cannot:
  - Time sharing
  - Priority scheduling

Real World Scheduling Analogies

- Which type of scheduling (preemptive / non-preemptive) occurs in the following settings?
  - Restaurant
  - Hospital emergency room
  - Professor’s office hours
Random OS Quote

➤ The two main design principles of the NeXT machine appear to be revenge and spite.

Don Lancaster

Scheduling Metrics

➤ There are a number of metrics that are commonly used to evaluate the performance of a scheduling algorithm:
  ✔ CPU Utilization
  ✔ Throughput
  ✔ Turnaround Time
  ✔ Wait Time
  ✔ Waiting Time
  ✔ Response Time

CPU Utilization

➤ CPU Utilization is the percentage of time that the CPU spends executing code on behalf of the users.
  ✔ Running user code
  ✔ Processing system calls
  ✔ Handling interrupts that signal completion of a requested operation.

Throughput / Turnaround Time

➤ Throughput is the average number of processes completed per time unit.
  ✔ E.g. 10 jobs / minute

➤ Turnaround time is the total time from when a process first enters the ready state to the last time it leaves the running state.
  ✔ Typically averaged across a number of jobs.

Wait Time / Waiting Time

➤ Wait time is the time a process spends in the ready queue before its first transition to the running state.

➤ Waiting time is the total time that a process spends in the ready queue during its entire execution.
  ✔ Both of these are typically reported as an average across a number of jobs.

Response Time

➤ Response time is the average length of a visit to the ready queue.
CPU Scheduling Algorithms

Scheduling Algorithms

- Basic Strategies:
  - First-Come-First-Served (FCFS)
  - Round Robin (RR)
  - Shortest Job Next (SJN)
  - Priority
- Combined Strategies:
  - Multi-level Queues
  - Multi-level Feedback Queues

Comparing Scheduling Algorithms

- There are a variety of techniques for evaluating and comparing the performance of different scheduling algorithms:
  - Deterministic Modeling
  - Implementation
  - Simulation
  - Theoretical approaches (e.g. Queuing models)

Modeling Processes

- In order to use deterministic modeling or simulation it is necessary to model how processes behave.
  - Typically processes alternate between bursts of CPU operations and blocking I/O requests.

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 10</td>
<td>CPU 100</td>
</tr>
<tr>
<td>I/O Disk1 200</td>
<td>I/O Disk1 100</td>
</tr>
<tr>
<td>CPU 20</td>
<td>CPU 100</td>
</tr>
<tr>
<td>I/O Disk1 150</td>
<td>I/O Disk1 100</td>
</tr>
<tr>
<td>CPU 10</td>
<td></td>
</tr>
</tbody>
</table>

Deterministic Modeling

- For deterministic modeling we will make the simplifying assumption that every process consists of only a single burst of CPU activity.
  - This allows the operation of the scheduler to be modeled more easily by hand.

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>CPU Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>15</td>
</tr>
<tr>
<td>P3</td>
<td>30</td>
</tr>
<tr>
<td>P4</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>CPU Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>350</td>
</tr>
<tr>
<td>P1</td>
<td>125</td>
</tr>
<tr>
<td>P2</td>
<td>475</td>
</tr>
<tr>
<td>P3</td>
<td>250</td>
</tr>
<tr>
<td>P4</td>
<td>75</td>
</tr>
</tbody>
</table>

FCFS Example

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>CPU Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>15</td>
</tr>
<tr>
<td>P3</td>
<td>30</td>
</tr>
<tr>
<td>P4</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>CPU Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>350</td>
</tr>
<tr>
<td>P1</td>
<td>125</td>
</tr>
<tr>
<td>P2</td>
<td>475</td>
</tr>
<tr>
<td>P3</td>
<td>250</td>
</tr>
<tr>
<td>P4</td>
<td>75</td>
</tr>
</tbody>
</table>

Gantt Chart:

Throughput = \( \frac{5 \text{ jobs}}{1275 \text{ ms}} = 0.004 \text{ jobs/ms} \)

Turnaround = \( \frac{(350 + (475 - 10) + (950 - 15) + (1200 - 30) + (1275 - 50)) \text{ ms}}{5 \text{ jobs}} = 929 \text{ ms} \)

Wait = \( \frac{(0 + (350 - 10) + (475 - 15) + (950 - 30) + (1200 - 50)) \text{ ms}}{5 \text{ jobs}} = 574 \text{ ms} \)
SJN Example

<table>
<thead>
<tr>
<th>CPU</th>
<th>Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>150</td>
</tr>
<tr>
<td>P1</td>
<td>125</td>
</tr>
<tr>
<td>P2</td>
<td>475</td>
</tr>
<tr>
<td>P3</td>
<td>250</td>
</tr>
<tr>
<td>P4</td>
<td>75</td>
</tr>
</tbody>
</table>

Gantt Chart:

\[
\text{Wait} = (0 + 75 + 450 + 800) \frac{tu}{5 \text{ jobs}} = 905 \frac{tu}{5 \text{ jobs}}
\]

\[
\text{Turnaround} = (75 + 200 + 450 + 800 + 1275) \frac{tu}{5 \text{ jobs}} = 560 \frac{tu}{5 \text{ jobs}}
\]

- SJN guarantees minimum average wait time.

Random CS Term

- The Ohnosecond
  - That minuscule fraction of time in which you realize you’ve made a big mistake.
  - `rm *.class`

Shortest Next CPU Burst Next

- The duration of past CPU bursts can be used as a predictor of the duration of the next CPU burst.
- One approach uses an exponential average:

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

- \(t_n\) = actual length of \(n^{th}\) (previous) CPU burst.
- \(\tau_n\) = predicted length of \(n^{th}\) CPU burst.
- \(\tau_{n+1}\) = predicted length of next CPU burst.
- \(\alpha\) = history scaling factor

Priority Scheduling

- With priority scheduling every process is assigned a priority value.
  - At each scheduling opportunity, the process with the highest priority is selected to run.
    - Priority scheduling can result in starvation.
    - Dynamic priorities and aging can be used to combat starvation.

Shortest Next CPU Burst Next

![Gantt Chart](image)

Priority Scheduling

![Gantt Chart](image)

RR Scheduling (w/ 50tu Time Slice)

<table>
<thead>
<tr>
<th>Arrival Order</th>
<th>CPU Burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>350</td>
</tr>
<tr>
<td>P1</td>
<td>125</td>
</tr>
<tr>
<td>P2</td>
<td>475</td>
</tr>
<tr>
<td>P3</td>
<td>250</td>
</tr>
<tr>
<td>P4</td>
<td>75</td>
</tr>
</tbody>
</table>

Gantt Chart:

\[
\text{Wait} = (0 + 50 + 100 + 150 + 200) \frac{tu}{5 \text{ jobs}} = 100 \frac{tu}{5 \text{ jobs}}
\]

\[
\text{Waiting}_{\text{total}} = (0 + 200 + 175 + 125 + 100 + 100 + 50) \frac{tu}{5 \text{ jobs}} = 750 \text{ tu}
\]

Response = \(\text{Waiting}_{\text{total}} + \text{Waiting}_{\text{total}}\) total visits to ready queue
RR Scheduling (w/ 10 tu scheduling overhead)

Gantt Chart:

<table>
<thead>
<tr>
<th>0</th>
<th>120</th>
<th>240</th>
<th>360</th>
<th>480</th>
<th>540</th>
<th>675</th>
<th>720</th>
<th>790</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
</tr>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P0</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
</tr>
</tbody>
</table>

CPU = \frac{(125 \times 125 + 475 \times 75) \text{tu}}{1535 \text{tu}} \times 100 = \frac{1275}{1535} \times 100 = 83\%

Throughput, turnaround, wait, waiting and response time calculations must now also include the overhead.

Multi-level Queues

- The ready queue is divided into multiple levels.
- The queue for each level has a different priority.
- Any job in higher priority queue will run before any job in lower priority queue.
- Each queue may use its own internal scheduling algorithm.
- Processes are assigned to a specific queue.

Multi-level Feedback Queues

- Similar to multi-level queues:
  - Each new process is assigned a priority.
  - The OS adjusts the priority of each process as it runs.
  - Thus, processes move among the different queues as they run.

Random OS Quote

- "The Internet? We are not interested in it"
  - Bill Gates, 1993
- "Sometimes we do get taken by surprise. For example, when the Internet came along, we had it as a fifth or sixth priority."
  - Bill Gates, 1998

Linux Scheduling (pre. V2.5)

- Early Linux kernels used preemptive scheduling with a multi-level queue with three levels:
  - FIFO: Highest priority level. Used for short, time-critical system threads.
  - RR: Medium priority. Used for longer running system threads.
  - OTHER: Lowest priority. Used for all user threads.
  - Internally, the OTHER queue uses a dynamic priority scheduling scheme.

Examples of Real OS CPU Scheduling Algorithms
Linux OTHER Queue Scheduling

- Each thread, \( i \), has a number of credits, \( p_i \).
  - New threads are given a default number of credits, \( K \).
- The system timer is used to create fixed size time slices.
  - On each timer interrupt the credits of the running thread is decremented.
  - If a thread’s credits reach 0, it is blocked.
- Scheduling is preemptive:
  - In each time slice, the thread in the ready queue that has the most credits is selected to run.
  - If no threads are ready (i.e. all threads are blocked), then a recrediting operation is performed.

Linux OTHER Recrediting

- If there are no threads ready to run recrediting occurs:
  - During recrediting, every thread in the system is assigned credits using the following formula:
    \[ p_i = \frac{p_i}{2} + K \]
  - Threads that were blocked because they had 0 credits now return to the ready queue with \( K \) credits.

Solaris

- Solaris uses preemptive scheduling with a multi-level queue with four levels.
  - The levels in order of decreasing priority are:
    - Real time: Provides guaranteed bound on response time.
    - System: Kernel threads
    - Interactive: User threads that are running in a windowing environment.
    - Time Sharing: Non-interactive user threads.

Solaris Interactive and Time Sharing Scheduling

- Priority within the interactive and time sharing queues is based on a dispatch table.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Slice Expired</th>
<th>Return from Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>160</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Windows XP Scheduling

- Windows XP uses preemptive scheduling with a multi-level queue with six levels.
  - Within each level, priority scheduling with seven relative priorities is used.
  - Priorities over 15 are fixed. Priorities less than 15 are dynamic and are adjusted based on process behavior.

<table>
<thead>
<tr>
<th>Base Priority</th>
<th>Fixed Priority</th>
<th>User Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Critical</td>
<td>31</td>
<td>26</td>
</tr>
<tr>
<td>Highest</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>Above Normal</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>Normal</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>Below Normal</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>Lowest</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Idle</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>

Windows XP Scheduling

- Windows XP scheduling does two unique things:
  - Priority Adjustments:
    - Like other schedulers:
      - Relative priority of thread using up its time slice is decreased.
      - Relative priority of thread returning from blocked is increased.
    - However, in XP the amount of the increase depends on why the thread was blocked.
  - Within the Normal priority class, threads in the foreground process get a longer time slice (3x) than other threads.
Multiprocessor Scheduling

- Some concepts in scheduling on multiprocessor / multicore machines:
  - Processor Affinity
    - Soft affinity
    - Hard affinity
    - Processor sets
  - Load Balancing
    - Push migration
    - Pull migration

Random OS Quote

- Here’s a few of 11 rules that Bill Gates gave to high school kids regarding stuff they won’t learn in school.
  - Rule 1: Life is not fair...get used to it.
  - Rule 4: If you think your teacher is tough, wait till you get a boss. He doesn't have tenure.
  - Rule 11: Be nice to nerds. Chances are you'll end up working for one.