Scheduling

Issue

• When a computer is multiprogrammed it frequently has multiple processes competing for the CPU at the same time.

• A choice has to be made which process to run next.

Other issues

• Depending on the application different scheduling strategies can make a difference.
  - Example:
    - Simple PC: networked server
    - Process switching is expensive.
    - User mode → kernel mode:
      - Save the state of current process
      - Run the scheduler
      - Load MMU
      - Run new program
      - The cache is now spoiled.

Process behavior

A spends most of his time computing; it is called compute-bound.
B spends most of his time waiting for I/O; it is called I/O-bound.

When to schedule

• A new process is created:
  - Select the new one or keep the current one running.

• A process terminates:
  - Select and run another process, if any.

• A process blocks (semaphore, I/O):
  - Dependencies between processes may improve scheduling.

• I/O interrupt:
  - Run a waiting process.

• Hardware interrupt:
  - Run the scheduler each clock interrupt or every k-th clock interrupt.
Scheduling can be divided:

- **non preemptive**
  - picks a process to run
  - lets it run until it blocks, terminates, or voluntarily releases the CPU
  - after a clock interrupt, resumes the process that was running before

- **preemptive**
  - picks a process to run
  - after a maximum amount of some fixed time suspends it (if still running)
  - picks another process to run (if any available)
  - requires clock

Scheduling: common goals

- **fairness**
  - comparable processes should get comparable service (CPU time)

- **policy enforcement**
  - different categories of processes may be treated differently

- **balance**
  - try to keep all parts of the system busy when possible

Scheduling: specific goals

- **batch systems**
  - throughput: # of processes completed per unit of time (hour)
  - turnaround time: average time to completion
  - CPU utilization

- **interactive systems**
  - response time (clear)
  - proportionality (with the difficulty of the task)

- **real-time systems**
  - meeting deadlines
  - predictability

Scheduling in Batch Systems (1)

- **First-Come First-Served**
  - nonpreemptive
  - the CPU is assigned in the order processes require it
  - when the running process blocks the following one in the queue is selected
  - when a blocked process becomes ready it is put on the end of the queue
  - simple (a single queue), fair
  - not optimal

- **Shortest Job First**
  - nonpreemptive
  - suppose we know the run-time in advance
  - the CPU is assigned to the shortest job in the queue
  - optimal if all the jobs are available at the same time

Example (1)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**turnaround:**
- A = 8
- B = 12
- C = 16
- D = 20
- average = 16

suppose a,b,c,d
- ta = a
- tb = a+b
- tc = a+b+c
- td = a+b+c+d

average = \( \frac{1}{4}(6a+3b+2c+d) \) ➔ shortest time first is optimal

Scheduling in Batch Systems (2)

- **Shortest Remaining Time Next**
  - preemptive (it is a preemptive version of the SJF)
  - the scheduler here chooses the process whose remaining run-time is the shortest
  - the time has to be known in advance
  - new short jobs get good service
Example (2)

compare with a preemptive algorithm

A, runs for 1s and blocks for I/O
B, C, D blocks after short time, they need to perform 1000 disk reads

B, C, D, take at least 1000s to complete

Scheduling in Interactive Systems (1)

• Round Robin
  each process is assigned a time interval, called quantum
  if the process is still running at the end of its quantum, the CPU is preempted and given to another process

Scheduling in Interactive Systems (2)

• Issues with Round Robin
  length of the quantum
  too short → context switch overhead
  too long → poor response to short interactive requests
  usually a reasonable value is 20-50 ms

• Priority Scheduling
  each process is assigned a priority
  priorities can be assigned:
  - statically
  - dynamically: e.g. assign more CPU to I/O bound processes
  divide processes in classes depending on priority
  use priority scheduling within classes
  round robin within classes

Example, 4 priority classes

<table>
<thead>
<tr>
<th>Priority</th>
<th>Process A</th>
<th>Process B</th>
<th>Process C</th>
<th>Process D</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
</tr>
<tr>
<td>3</td>
<td>P5</td>
<td>P6</td>
<td>P7</td>
<td>P8</td>
</tr>
<tr>
<td>2</td>
<td>P9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: dynamic priority

assign priority depending on the fraction of quantum each process has used

\[ P = \frac{1}{f} \]

example: time slice 50ms
process A uses 1ms, f=1/50, priority = 50
process B uses 50 ms, f=50/50, priority = 1

Scheduling in Interactive Systems (3)

• Shortest Process Next
  shortest job produces the minimum average response time for batch systems
  the problem here is figuring which of the runnable processes is the shortest one
  solution: use estimates based on past behavior
  \[ \hat{T} = aT_i + (1-a)\hat{T}_{i-1} \]
  Example: \( a = 0.5 \)

\[ \hat{T}_1 = \frac{\hat{T}_1}{2}\]
\[ \hat{T}_2 = \frac{\hat{T}_2}{2}\]
\[ \hat{T}_3 = \frac{\hat{T}_3}{4}\]
\[ \hat{T}_4 = \frac{\hat{T}_4}{8}\]
\[ \hat{T}_5 = \frac{\hat{T}_5}{4}\]
\[ \hat{T}_6 = \frac{\hat{T}_6}{2}\]

\[ \hat{T}_7 = \frac{\hat{T}_7}{4}\]
\[ \hat{T}_8 = \frac{\hat{T}_8}{8}\]
\[ \hat{T}_9 = \frac{\hat{T}_9}{4}\]
\[ \hat{T}_{10} = \frac{\hat{T}_{10}}{2}\]
Scheduling in Interactive Systems (4)

• Guaranteed Scheduling
make promises about performance to the users/processes
compute the real amount of CPU a user/process has consumed
increase priority accordingly
difficult to implement

• Lottery Scheduling
basic idea: give processes lottery tickets for various system resources (CPU time)
whenever a scheduling decision is required a lottery ticket is randomly chosen
similar to priority scheduling, but:
- the rule is clearer
- interesting properties: tickets can be exchanged (a process/user can own/trade tickets)

Policy versus Mechanism

• Often a process has many children running under its control performing different tasks. In this case only the process itself knows which one is the most important or time critical
• For this reason it is important to separate scheduling mechanism from the scheduling policy
• The scheduling mechanism (algorithm) defines the parameters used by the scheduler
• The user process is responsible for filling in those parameters for its children (policy)

Scheduling in Real Time Systems

In real time systems time plays a crucial role. Usually the system is connected to one or more external devices which generate stimuli and the OS has to react appropriately to them within a fixed amount of time.
Examples: aircraft control, over-temperature monitor in nuclear power station, ABS, biomedical systems, robotics
• hard-real time, missing a deadline has catastrophic effects
• soft-real time, missing a deadline is undesirable but tolerable
Stimuli (events) may be:
• periodic (occurring at regular intervals)
• aperiodic (unpredictable)

Schedulability

• Depending on the situation, it may happen that not all the events can be handled
• Consider $m$ periodic events
  event $i$ occurs with period $P_i$ and requires $C_i$ second of CPU time
  the system is schedulable if:
  $$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Let’s consider the following situation:

Multimedia system: three processes A, B, C
A is periodic, $T = 30$ ms, and uses 10 ms of CPU time
B is periodic, $f = 25$ Hz ($T = 40$ ms) and uses 15 ms of CPU time
C is periodic, $f = 20$ Hz ($T = 50$ ms) and uses 5 ms of CPU time
Schedulability: $10/30 + 15/40 + 5/50 = 0.808 < 1$
Rate Monotonic Scheduling (RMS)

- Assumptions:
  - each periodic process must complete within its period
  - no process is dependent on any other process
  - each process needs the same amount of CPU time on each burst
  - any non-periodic processes have no deadlines
  - preemption has no overhead
- Assign each process a fixed (static) priority equal to the frequency of occurrence of its triggering event (priorities are linear with the rate)

Earliest Deadline First Scheduling (EDF)

- Assumptions:
  - the same as rate monotonic but
  - it doesn’t require processes to be periodic
  - processes can use different amounts of CPU for different bursts
- Runnable processes are kept in a list with their deadline
- The scheduler runs the process with the closest deadline
- Preempts the current process if another one with a closer deadline is ready

RMS versus EDF

RMS uses static priorities and fails if CPU utilization is too high. EDF always works if CPU utilization is < 100%

Case study: scheduling in win32

- Only threads are scheduled, not processes
- Time-sliced, round robin with priorities
- Threads have priorities 0 through 31
How are priorities assigned?

WinMax Process Classes

<table>
<thead>
<tr>
<th>WinMax Thread Priorities</th>
<th>Real Time</th>
<th>High</th>
<th>Above Normal</th>
<th>Normal</th>
<th>Below Normal</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-critical</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Highest</td>
<td>26</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Above-normal</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Normal</td>
<td>24</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Below-Normal</td>
<td>23</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Lowest</td>
<td>22</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Idle</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Priority Boost

- dynamic boost (< 15)
  - foreground threads get doubled time slice
  - if resumed by keyboard/mouse + 6
  - if resumed on wait +1
- decay: after boost priority is reduced of one level until it reaches base priority (the priority before boost)

CPU Starvation

- Balance Set Manager (priority 16, every second)
  - looks for “starved thread” that have been ready for more than 4 seconds
- Special boost:
  - set priority to 15
  - doubled quantum
- Apply only to non real-time threads