Real Time Systems

- Program must execute within strict time constraints
- Often *embedded system*—program/computer is part of larger system
- Time may be a parameter in computation (e.g., sampling quantities)
- Often RT system will also include non-RT tasks
- Predictability is more important than speed

Synchronous scheduling (clock driven)
- Processor time divided into fixed duration *frames*
- Divide program into segments that can be completed in a single frame
- Static schedule assigns segments (possibly more than one) to frames
- Segment is only started if it will complete (worst case timing) before end of frame
- Performance guaranteed
- Well suited to continuous, periodic tasks
- Wastage: unused processor time at end of frames
- Periodicity limited to multiples of frame size
- Schedule is very difficult, error-prone and system dependent.

Asynchronous scheduling (interrupt driven)
- Processes (segments) execute to completion
- Scheduler uses priority (or deadlines) to decide order of execution
- *pre-emptive scheduling*—an executing process can be interrupted—pre-empted—to allow a higher priority process to execute.
- Related to *time-slicing*—all processes have the same priority. Periodic pre-emption (task switching).
- Possible to get 100% processor utilization
- Overall faster processing
- Performance is dependent on other (higher priority) processes

Mixed Sync./Async. Scheduling
- Synchronous scheduling of time-critical tasks
- Asynchronous scheduling to fill in gaps (background processes)

Input and Output
- Synchronous is good for devices that require polling
- Asynch. for devices that generate interrupts
Priority Scheduling

Assume periodic tasks. period of $\tau_i = T_i$, duration = $d_i$.

**response time**: delay between request to execute and completion

**overflow**: when a task must execute another cycle before previous one has completed (i.e., response time > $T_i + d_i$)

**feasible priority assignment**: no overflow

Example: $T_1 = 2$, $T_2 = 5$, $d_1 = 1$, $d_2 = 2$
Feasible assignment: $P_1 > P_2$
Infeasible assignment: $P_2 > P_1$

**Theorem** Longest response time occurs when request corresponds to all higher priority requests.

Earliest Deadline First

Dynamically assign priority based on closest deadline (time of next request)

Feasible iff $\sum_i \frac{d_i}{T_i} \leq 1$

Rate Monotonic Scheduling

Assign priority in decreasing order of intervals between requests, i.e., $T_i < T_j \Rightarrow P_i > P_j$

• Will give a feasible assignment if one exists.
• May waste time.
• Based on fixed duration and repetition rates.

Priority Inversion

Consider a set of jobs, $J_1, J_2, \ldots J_n$ s.t. $J_1$ is highest priority and $J_n$ is lowest.

Assume

• A job will not suspend itself
• Critical sections in job are properly nested
• Job will release all locks on completion

**periodic tasks**—sequence of the same type of jobs that must be executed at regular intervals

**aperiodic tasks**—sequence of the same type of jobs that are executed at irregular intervals (e.g., in response to input)
• Each task, $\tau_i$, has fixed priority $P_i$.

• Initially jobs have same priority as the task that contains them

• If several jobs are eligible to run, run the highest priority

• Jobs with same priority executed in FCFS order.

Priority inversion: Higher priority process is blocked by lower priority process.

Simple example,

• $J_1$ and $J_2$ have mutual critical sections.

• $J_2$ reaches critical section first—$J_1$ will be blocked waiting for $J_2$.

Another example

• $J_1$ is blocked trying to synchronize with $J_3$

• $J_2$ gets to execute, preventing $J_3$ from executing

• $J_1$ is waiting for $J_2$ (arbitrarily long)

Non-preemptable Critical Sections

• CS must be short

• Results in unnecessary blocking:
  - $J_3$ enters CS
  - $J_1$ is blocked, even if it doesn’t want to enter its CS (assuming uniprocessor)

Monitors

• Make monitor higher priority than all callers

• Low priority caller can block higher priority caller

Priority Inheritance

• Each job uses its assigned priority, unless it is in a critical section and blocks higher priority jobs.

• $J$ inherits the highest priority of the jobs blocked by $J$.

• When $J$ exits critical section, priority set back to $P$ at entry to CS.

• Inheritance is transitive.

• Priority change operations are atomic.

Guarantees upper bound on total blocking delay (assuming no deadlock).

Problems

1) Can deadlock.

2) Blocking duration can be long.
Priority Ceiling Protocol

A job in its CS will execute with priority higher than inherited priorities of all other preempted CS.

- Assign *priority ceiling* to semaphores = highest priority task that may use it
- $J_i$ can start CS only if $P_i >$ priority ceiling for all semaphores locked by other jobs.

---

RMA Example 1

<table>
<thead>
<tr>
<th>Task</th>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$U_i$</th>
<th>$\sum U_i$</th>
<th>$S_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>25</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>100</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>200</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>500</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Fig. 1. Sequence of events described in Example 2.

Fig. 2. Sequence of events described in Example 4.
### RMA Example 2

<table>
<thead>
<tr>
<th>Task</th>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$U_i$</th>
<th>$\sum U_i$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>25</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>90</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>140</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>500</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Blocking

Assume priority ceiling protocol: $B_i = $ longest time a job may be blocked (max duration of CS of lower priority job guarded by semaphore with priority ceiling $> P_i$).

<table>
<thead>
<tr>
<th>Task</th>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$B_i$</th>
<th>$U_i$</th>
<th>$\sum U_i$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>25</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>100</td>
<td>30</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>200</td>
<td>40</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>500</td>
<td>100</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>