Real-Time Computing

Definition of Real-Time Computing

- Real-time computing is computing in which the time of the computation plays an essential role in the result.
- This includes:
  - Computations that measure time
  - Computations that must meet deadlines
  - Computations that synchronize other computations based on time

Example of Real-Time
(from Briand and Roy, Meeting deadlines in hard real-time systems, IEEE Press, 1999)

- July 20, 1969, landing module 10k feet above the Moon:
  - Houston: “Eagle, you’re go for a landing.”
  - Houston: “One minute [of fuel left].”
  - Lander: “100 feet, 3 1/2 down, 9 forward.”
  - Houston: “30 Seconds.”
  - Lander: “OK, engine stop.”

Example of Real-Time
(from Briand and Roy, Meeting deadlines in hard real-time systems, IEEE Press, 1999)

- “During the descent of the Eagle [landing module], an incorrect switch position caused the analog-to-digital conversion circuit of the rendezvous to send some bursts of high-priority requests to the computer. After 15 percent of the computer resources were tied up in responding to the spurious requests, jobs began to miss their deadlines. A hardware recovery mechanism detected the timing fault and restarted the computer.”

Distinctions

- Hard real-time: Deadlines must be met in order for the system to be correct.
- Soft real-time: It is desirable for deadlines to be met, but if not, the system can still be correct.
- “Real-time” does not necessarily mean “fast”.

Hard Real-Time Examples

- Vehicular fuel or rendezvous problems
  - Lunar lander
  - Train scheduling
  - Baggage handlers
  - Assembly lines
- Production deadlines
  - Newspaper
  - Live TV show
  - Graduation
Hardness Expressed as a Utility Function

Progressive Utility

- A generalization of the preceding concept regards Utility as a function of both time and fraction of the task completed.

Real-Time includes Communication

- Obviously communication, as well as computation, must be taken into account if the system consists of multiple components

Real-Time Operating Systems

- Solaris, when generated with real-time features
- VxWorks (Wind River Systems, Inc.)

Common Aspects of Real-Time Systems

- Deadlines
- Scheduled task start times
- Timeouts
- Periodic & Aperiodic tasks
- Priorities among tasks

Scheduling Algorithms

- Choices of performance metric to meet requirement:
  - Total completion time among all tasks
  - Average response time over all tasks
  - Weighted sum of completion times
  - Maximum lateness
  - Number of late tasks
Scheduling to Meet Deadlines

- Assume a set of tasks \( \{T_i\} \)
- Associate with each task \( T_i \):
  - computation time \( C_i \)
  - deadline \( D_i \)
- Suppose we want to minimize maximum lateness

Proof that Jackson’s Rule works

- Let \( S' \) be a schedule that executes the tasks in some order other than by Jackson’s Rule.
- Let \( S \) be a schedule that executes a pair of tasks out-of-order in \( S' \) in order of decreasing deadline.
- We want to show that the maximum lateness of \( S \) is no more than that of \( S' \)

Proof of Jackson’s Rule

- Case \( L_a < L_b \):
  \[
  \max(L_a', L_b') = L_b \\
  = F_b - D_b \quad \text{defn of L} \\
  = F_a' - D_b \quad \text{a and b are flipped in } S' \text{ vs } S \\
  \leq F_a' - D_a \\
  = L_a' \\
  \leq \max(L_a', L_b')
  \]
Proof of Jackson’s Rule

- Case $L_a \geq L_b$:
  \[
  \max(L_a, L_b) = L_a = F_a - D_a = L'_a < F'_a - D_a = L'_b \leq \max(L'_a, L'_b)
  \]
  (def'n of max)

Corollary to Jackson’s Rule

- Assume that tasks are numbered in order of increasing deadlines $D_i$.
- Let $C_i$ be the corresponding computation time.
- Then all tasks can be executed so as to meet their deadline provided that for all $i$
  \[
  \sum_{k=1}^{i} C_k \leq D_i
  \]

Limitation of Jackson’s Rule

- The set of all tasks is not always presented in advance.
- New tasks may arrive at arbitrary times.
- To minimize maximum lateness in this setting, it may be necessary to preempt a task already being executed.
- This issue is addressed by Horn’s rule.

Horn’s Rule (1974)

- Arrange execution, using preemption if necessary, so that:
  At every instant the task with the current earliest deadline is being executed.
- Horn’s rule can be proved similar to Jackson’s.
Horn's Rule (1974)

- If preemption is not allowed, then Horn's rule does not minimize maximum lateness.
- Example:

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
<th>Deadline</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>4</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>T₂</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Horn's rule says: start T₁ at time 0, which would make T₂ wait to time 4. The max lateness would be 1.

The optimum way would be do nothing at time 0, start T₂ at time 1, then start T₁ at time 3. The maximum lateness would be 0.

Exercise

- What happens in the previous example if preemption is allowed?

Hard Problem

- Finding an optimal non-preemptive schedule when arrival times are arbitrary is NP-hard.
- An enumerative, branching, algorithm can be used.

Lawler's Algorithm (1973)

- Schedules a set of simultaneously arriving tasks on one process subject to precedence constraints.
- Minimizes maximum lateness for 1 processor, non-preemptive.

Lawler's Algorithm (1973)

- Build a stack, selecting tasks with latest deadline first (LDF), subject to precedence constraints.
- Execute the tasks in order of popping from the stack.
Example: Lawler's Algorithm

Assume all computation times are 1 and deadlines are as shown.

Stacking order (LDF, obeying precedence): T_6 /6, T_5 /5, T_3 /4, T_4 /3, T_2 /5, T_1 /2

Execution order: T_1 /2, T_2 /5, T_4 /3, T_3 /4, T_5 /5, T_6 /6

Completion: 1 2 3 4 5 6

Using EDF, the order would be: T_1 /2, T_3 /4, T_2 /5, T_4 /3, T_5 /5, T_6 /6
Completion: 1 2 3 4 5 6

One task would be late, so max. lateness = 1.

Other Algorithms

- An optimal method for the preemptive arbitrary-arrival case is also known (run time $O(n^2)$).
- This and the previous methods may be found in reference:
  - G.C. Buttazzo
  Hard Real-Time Computing Systems

Periodic Tasks

- Many realtime systems are based on periodic tasks, where each task T_i has:
  - Computation time C_i
  - Period P_i
  - Phase $\phi_i$
- The meaning of “period” is that, for each i, T_i must execute once every time P_i units.
- The meaning of “phase” is that, for each i, the earliest time at which T_i is available within the current P_i period is at relative time $\phi_i$. 
Priority, Periodicity, and RMA

- Assume that tasks are preemptable.
- A task that is preempted by another is said to have lower priority.
- Obviously tasks with shorter periods should generally have higher priority, since lower priority tasks can be preempted, then resumed, to allow higher priority tasks to meet their periodic deadlines.
- Since shorter period $\Rightarrow$ higher rate, this is called rate-monotonic analysis (RMA).

Preemption Costs

- In general, there will be a cost (delay) associated with preemption of a task.
- For now, we assume that this cost is negligible.

Note on RMA Assumption

- RMA is a mathematical notion.
- It should not be inferred that every set of user priorities will agree with RMA.

Example of RMA

- Three periodic tasks:
  
<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
<th>Period</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>0.5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$T_2$</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$T_3$</td>
<td>1.75</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
  
- For example, since $T_1$ has period 10 and phase 3, it is available at 3, 13, 23, 33, ….

Exercise

- Given the previous table, construct a schedule that schedules the tasks periodically.
- Remember that tasks can be preempted.