Introduction to Real-Time Systems

Note: Slides are adopted from Lui Sha and Marco Caccamo
Overview

- Today: this lecture introduces real-time scheduling theory

- To learn more on real-time scheduling terminology:
  - see chapter 4 see basic concepts on “Hard Real-Time Computing Systems” book from G. Buttazzo

- Basic tutorial at:
  http://www.embedded.com/electronics-blogs/beginner-s-corner/4023927/Introduction-to-Rate-Monotonic-Scheduling#
So What is a Real-Time System?

- A **real-time system** is a system whose specification includes both logical and temporal correctness requirements.

  ✓ **Logical Correctness**: produces correct outputs.

  ✓ **Temporal Correctness**: produces outputs at the right time.
### So What is a Real-Time System?

A real-time system has different set of measure of merits:

<table>
<thead>
<tr>
<th></th>
<th>Real time systems</th>
<th>Non-real time systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>performance</strong></td>
<td>Schedulable Utilization (schedulability)</td>
<td><strong>throughput</strong></td>
</tr>
<tr>
<td><strong>responsiveness</strong></td>
<td>Worst case response time of each tasks</td>
<td><strong>Average response time</strong></td>
</tr>
<tr>
<td><strong>overload</strong></td>
<td>Stability (getting critical tasks done)</td>
<td><strong>Fairness</strong></td>
</tr>
</tbody>
</table>
What does *Real-Time* mean?

- The word *time* means that the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced.

- The word *real* indicates that the reaction of the system to external events must occur during their evolution. As a consequence, the system time (internal time) must be measured using the same time scale used for measuring the time in the controlled environment (external time).

  [in chapter 1 of Buttazzo’s book]
Real-Time systems

- Advances in computer hardware will not take care of the temporal requirements needed by a real-time system.
  - The old “buy a faster processor” argument does not work!
  - An old Pentium can be used to run a real-time application
  - A last generation pc with a general purpose operating system (windows, linux, etc.) can violate the temporal constraints of our real-time application.

- Rather than being fast, a real-time computing system should be predictable.
Are All Systems Real-Time systems?

- Question: is a payroll processing system a real-time system?
  - It has a time constraint: print the pay checks every two weeks

- Perhaps it is a real-time system in a definitional sense, but it doesn’t pay us to view it as such.

- We are interested in systems for which it is not a priori obvious how to meet timing constraints.
**Typical Real-Time Systems**

- Cell phones, digital cameras
- Avionic
- Radar Systems
- Factory Process control
- Sensing and Control
- Multi-media systems

- All of them have explicit timing requirements to meet.
Jobs and Tasks

- A job is a unit of computation, e.g.,
  - handling the press of a keyboard
  - or compute the control response in one instance of a control loop

- A task is a sequence of the same type of jobs, say, a control task or the keyboard handling task.
Periodic Task Model

- Periodic tasks are the “work horse” of real-time systems and they play a key role in real-time systems.

- A task $\tau_i$ is said to be periodic if its inter-arrival time (period), $p_i$, is a constant

- A periodic task, $\tau_i$, is characterized by
  - Period, $p_i$
  - Release time, $r_{i,j}$. The default is $r_{i,j} = r_{i,j-1} + p_i$
  - Execution, $C_i$. The default is worst-case execution time
  - Relative deadline $D_i$. The default is equal to period
  - Phase $\phi_i$: the starting time of the task, i.e., the first release time ($r_{i,1}$).
Release Time and Deadlines

- Release time is the instant at which the job becomes ready to execute.
- The common form of deadlines are absolute deadlines where deadlines are specified in, well, absolute times. Train and airlines schedules have absolute deadlines.
- Normally, relative deadlines are related to the release time. For example, a relative deadline $D=8$ msec after the release time.
- The default absolute deadline of a task is the end of period. By convention, we will refer to an absolute deadline as “$d$”, and a relative deadline as “$D$".

![Diagram showing release time and deadlines](image)
A Sample Problem

Periodic tasks

- $\tau_1$: 100 msec (period)
  - 20 msec
- $\tau_2$: 150 msec (period)
  - 40 msec
- $\tau_3$: 350 msec (period)
  - 100 msec

Shared resources

- shared data1: 2 msec
- shared data2: 10 msec

Aperiodics

- Emergency: 50 msec (min interarrival time)
  - 5 msec
- Deadline: 6 msec after arrival
- Non-critical display: 40 msec (avg interarrival time)
  - 2 msec

Desired response: 20 msec average

Goal: guarantee that no real-time deadline is missed!!!
Real-time scheduling algorithms

- Jobs can be scheduled according to the following scheduling algorithms:
  
  - **Rate Monotonic (RM)**: the faster the rate, the higher is the priority. All the jobs in a task have the same priority and hence the name “fixed priority” algorithm.
  
  - **Earliest Deadline First (EDF)**: the job with the earliest deadline has the highest priority. Jobs in a task have different priorities and hence the name, “dynamic priority” algorithm.
Priority and Criticality - 1

- **Priority**: priority is the order we execute ready jobs.
- **Criticality (Importance)**: represents the penalty if a task misses a deadline (one of its jobs misses a deadline).

- **Quiz**: Which task should have higher priority?
  - **Task 1**: The most important task in the system: if it does not get done, serious consequences will occur
  - **Task 2**: A mp3 player: if it does not get done in time, the played song will have a glitch

- If it is feasible, we would like to meet the real-time deadlines of both tasks!
Priority and Criticality - 1

- Priority: priority is the order we execute ready jobs.
- Criticality (Importance): represents the penalty if a task misses a deadline (one of its jobs misses a deadline).

- Quiz: Which task should have higher priority?
  - Answer: the task with higher rate (according to RM) unless the system is overloaded!

- Task 1: The most import task in the system: if it does not get done, serious consequences will occur
- Task 2: A mp3 player: if it does not get done in time, the played song will have a glitch

- If it is feasible, we would like to meet the real-time deadlines of both tasks!
If priorities are assigned according to importance, there is no lower bound of processor utilization, below which tasks deadlines can be guaranteed.

\[
\frac{C_1}{p_1} + \frac{C_2}{p_2} = U
\]

\[
U \to 0, \text{ when } C_2 \to 0 \text{ and } p_1 \to \infty
\]

Task \( T_2 \) will miss its deadline, as long as \( C_1 > p_2 \).
Priority and Criticality - 3

- An important find in real-time computing theory is that importance may or may not correspond to scheduling priority.
- In the previous example, giving the less important task higher priority results in both tasks meeting their deadlines.
- Importance matters only when tasks cannot be scheduled (overload condition) without missing deadlines.

![Diagram showing important and less important jobs]
Utilization and Schedulability

• A task’s utilization (of the CPU) is the fraction of time for which it uses the CPU (over a long interval of time).

• A periodic task’s utilization $U_i$ (of CPU) is the ratio between its execution time and period: $U_i = \frac{C_i}{p_i}$

• Given a set of periodic tasks, the total CPU’s utilization is equal to the sum of periodic tasks’ utilization:
  \[ U = \sum_{i} \frac{C_i}{p_i} \]

• Schedulability bound of a scheduling algorithm is the percentage of CPU utilization at or below which a set of periodic tasks can always meet their deadlines. You may think of it as a standard benchmark for the effectiveness of a scheduling algorithm.

• QUIZ: What is the obvious limit on $U$?
Utilization and Schedulability

- A task’s utilization (of the CPU) is the fraction of time for which it uses the CPU (over a long interval of time).

- A periodic task’s utilization $U_i$ (of CPU) is the ratio between its execution time and period: $U_i = C_i/p_i$

- Given a set of periodic tasks, the total CPU’s utilization is equal to the sum of periodic tasks’ utilization:

$$U = \sum_{i} \frac{C_i}{p_i}$$

- Schedulability bound of a scheduling algorithm is the percentage of CPU utilization at or below which a set of periodic tasks can always meet their deadlines. You may think of it as a standard benchmark for the effectiveness of a scheduling algorithm.

- QUIZ: What is the obvious limit on $U$?
  - ANSWER: 1, you cannot utilize more than 100% of the processor capacity!
Real-time scheduling algorithms

- Scheduling algorithms need to be simple: cannot use many processor cycles

- Static vs. dynamic priorities
  - Static priority: All jobs of a task have the same priority
  - Dynamic priority: Different jobs of the same task may have different priorities

- Examples
  - Rate Monotonic Scheduling [RM]: Tasks with smaller periods are assigned higher priorities (static priority)
  - Earliest Deadline First [EDF]: Jobs are prioritized based on absolute deadlines (dynamic priority)
Dynamic vs “Static” Priority Scheduling in Theory

- An instance of a task is called a job. “Static” priority assigns a (base) priority to all the jobs in a task. Dynamic priority scheduling adjusts priorities in each task job by job.

- Quiz: what type of scheduling algorithm is used to schedule these two tasks?

- An optimal dynamic scheduling algorithm is the earlier deadline first (EDF) algorithm. Jobs closer to deadlines will have higher priority. With independent periodic tasks, all tasks will meet their deadlines as long as the processor utilization is less than or equal to 1.

- An optimal “static” scheduling algorithm is the rate monotonic scheduling (RMS) algorithm. For a periodic task, the higher the rate (frequency) the higher the priority.
Dynamic vs “Static” Priority Scheduling in Theory

- An instance of a task is called a job. “Static” priority assigns a (base) priority to all the jobs in a task. Dynamic priority scheduling adjust priorities in each task job by job.

- Quiz: what type of scheduling algorithm is used to schedule these two tasks?
- Answer: EDF

- An optimal dynamic scheduling algorithm is the earlier deadline first (EDF) algorithm. Jobs closer to deadlines will have high priority. With independent periodic tasks, all tasks will meet their deadlines as long as the processor utilization is less than 1.

- An optimal “static” scheduling algorithm is the rate monotonic scheduling (RMS) algorithm. For a periodic task, the higher the rate (frequency) the higher the priority.
Importance of the scheduling algorithm

- To demonstrate the importance of a scheduling algorithm, consider a system with only two tasks, T1 and T2. Assume these are both periodic tasks with periods $p_1$ and $p_2$, and each has a deadline that is the beginning of its next cycle.

- Task 1 has $p_1 = 50$ ms, and a worst-case execution time of $C_1 = 25$ ms. Task 2 has $p_2 = 100$ ms and $C_2 = 40$ ms. Note that the utilization, $U_i$, of task $i$ is $C_i/T_i$. Thus $U_1 = 50\%$ and $U_2 = 40\%$. This means total requested utilization $U = U_1 + U_2 = 90\%$. It seems logical that if utilization is less than 100\%, there should be enough available CPU time to execute both tasks.

- Let's consider a static priority scheduling algorithm. With two tasks, there are only two possibilities:
  - Case 1: Priority(T1) > Priority(T2)
  - Case 2: Priority(T1) < Priority(T2)

- The two cases are shown in next figure. In Case 1, both tasks meet their respective deadlines. In Case 2, however, Task 1 misses a deadline, despite 10\% idle time. This illustrates the importance of priority assignment.
Importance of the scheduling algorithm

Both possible outcomes for static-priority scheduling with two tasks ($T1=50$, $C1=25$, $T2=100$, $C2=40$)

See: http://www.netrino.com/Publications/Glossary/RMA.html
Importance of the scheduling algorithm

- It is theoretically possible for a set of tasks to require just 70% CPU utilization in sum and still not meet all their deadlines. For example, consider the case shown in the Figure below. The only change is that both the period and execution time of Task 2 have been lowered. Based on RM, Task 1 is assigned higher priority. Despite only 90% utilization, Task 2 misses its first deadline. Reversing priorities would not have improved the situation.

Some task sets aren't schedulable \((T1=50, C1=25, T2=75, C2=30)\)

Exercise: try to use EDF and check if the first deadlines are met!
Key scheduling results

- For periodic tasks with relative deadlines equal to their periods:
  - Rate monotonic priority assignment is the optimal static priority assignment
    - No other static priority assignment can do better
    - Yet, it cannot achieve 100% CPU utilization
- Earliest deadline first scheduling is the optimal dynamic priority policy
  - EDF can achieve 100% CPU utilization
- More details in the next lecture