Real-Time Systems

- Definition
  - Systems whose correctness depends on their temporal aspects as well as their functional aspects

- Performance measure
  - Timeliness on timing constraints (deadlines)
  - Speed/average case performance are less significant.

- Key property
  - Predictability on timing constraints

Real-time Systems

- Real-time monitoring systems
- Signal processing systems (e.g., radar)
- On-line transaction systems
- Multimedia (e.g., live video multicasting)
- Embedded control systems:
  - automobiles
  - Robots
  - Aircrafts
  - Medical devices...

Real-Time System Example

- Digital control systems
  - periodically performs the following job:
    - *senses* the system status and
    - *actuates* the system according to its current status

Scheduling Framework Example

- Multimedia applications
  - periodically performs the following job:
    - reads, decompresses, and displays video and audio streams
**Fundamental Real-Time Issue**

- To specify the timing constraints of real-time systems
  - Hard temporal constraints
  - Soft temporal constraints
- To achieve predictability on satisfying their timing constraints, possibly, with the existence of other real-time systems

**Soft Temporal Constraints**

- A soft real-time system is one where the response time is normally specified as an average value. This time is normally dictated by the business or market.
- A single computation arriving late is not significant to the operation of the system, though many late arrivals might be.
- Ex: Airline reservation system - If a single computation is late, the system's response time may lag. However, the only consequence would be a frustrated potential passenger.

**Hard Temporal Constraints**

- A hard real-time system is one where the response time is specified as an absolute value. This time is normally dictated by the environment.
- A system is called a hard real-time if tasks always must finish execution before their deadlines or if message always can be delivered within a specified time interval.
- Hard real-time is often associated with safety critical applications. A failure (e.g., missing a deadline) in a safety-critical application can lead to loss of human life or severe economical damage.

**Real-Time Spectrum**

- No RT
- Soft RT
- Hard RT

**Real-Time Workload**

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters

**Real-Time Task**

- Task: a sequence of similar jobs
  - Periodic task \((p,e)\)
    - Its jobs repeat regularly
    - Period \(p\) = inter-release time \((0 < p)\)
    - Execution time \(e\) = maximum execution time \((0 < e < p)\)
    - Utilization \(U = e/p\)
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines

Real-Time Scheduling

- Determines the order of real-time task executions

Real-Time Scheduling

- Static scheduling
  - A fixed schedule is determined statically
  - E.g., Cyclic Executive
- Static-priority scheduling
  - Assign fixed priorities to processes
  - A scheduler only needs to know about priorities
  - E.g., Rate Monotonic (RM)
- Dynamic-priority scheduling
  - Assign priorities based on current state of the system
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period

RM (Rate Monotonic)

- Executes a job with the shortest period

Deadline Miss!
Response Time

- Response time: Duration from released time to finish time

RM - Schedulability Analysis

- Real-time system is schedulable under RM if and only if \( r_i \leq p_i \) for all task \( T_i(p, e_i) \)

Joseph & Pandya,
"Finding response times in a real-time system",

RM – Utilization Bound

- Real-time system is schedulable under RM if \( \sum U_i \leq n (2^{1/n} - 1) \)

Liu & Layland,
"Scheduling algorithms for multi-programming in a hard-real-time environment",

Example: \( T_1(4,1), T_2(5,1), T_3(10,1) \),
\[
\sum U_i = 1/4 + 1/5 + 1/10 = 0.55
\]
\(3 (2^{1/3} - 1) \approx 0.78\)
Thus, \( \{T_1, T_2, T_3\} \) is schedulable under RM.
RM – Utilization Bound

- Real-time system is schedulable under RM if
  \[ \sum U_i \leq n \left( \frac{2^{1/n}}{-1} \right) \]

**RM Utilization Bounds**

EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline

EDF (Earliest Deadline First)

- Executes a job with the earliest deadline

EDF (Earliest Deadline First)

- Executes a job with the earliest deadline

EDF (Earliest Deadline First)

- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it.
Processor Demand Bound

- Demand Bound Function: \( \text{dbf}(t) \)
  - the maximum processor demand by workload over any interval of length \( t \)

EDF - Schedulability Analysis

- Real-time system is schedulable under EDF if and only if \( \text{dbf}(t) \leq U \) for all interval \( t \)


- Demand Bound Function: \( \text{dbf}(t) \)
  - the maximum processor demand by workload over any interval of length \( t \)

EDF – Utilization Bound

- Real-time system is schedulable under EDF if and only if

\[ \sum U_i \leq 1 \]


EDF – Overload Conditions

- Domino effect during overload conditions
  - Example: \( T_1(4,3), T_2(5,3), T_3(6,3), T_4(7,3) \)

Deadline Miss!

Better schedules:

RM vs. EDF

- Rate Monotonic
  - Simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines)
  - Predictability for the highest priority tasks
- EDF
  - Full processor utilization
  - Misbehavior during overload conditions

For more details: Buttazzo, "Rate monotonic vs. EDF: Judgement Day", EMSOFT 2003.

Priority Inversion and the MARS Pathfinder

- Landed on the Martian surface on July 4th, 1997
  - Unconventional landing - bounching onto the Martian surface
- A few days later, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system reset, each resulting in losses of data

What happened:
- Pathfinder has an "information bus"
- The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data (while holding the mutex on bus)
- A communication task that ran with medium priority
- It is possible for an interrupt to occur that caused (medium priority) communications task to be rescheduled during the time the (low priority) meteorological data thread was blocked waiting for the bus.
- After some time passed, a watch dog timer goes off, noticing that the data bus has not been executed for some time, it concluded that something had gone really bad, and initiated a total system reset.
The Priority Inversion Problem

Priority order: T1 > T2 > T3

T1

failed attempt to lock R

lock(R)

unlock(R)

T2

lock(R)

unlock(R)

T3

lock(R)

unlock(R)

T2 is causing a higher priority task T1 wait!

Priority Inversion

1. T1 has highest priority, T2 next, and T3 lowest
2. T3 comes first, starts executing, and acquired some resource (say, a lock).
3. T1 comes next, interrupts T3 as T1 has higher priority
4. But T2 needs the resource locked by T3, so T1 gets blocked
5. T3 resumes execution (this scenario is still acceptable so far)
6. T2 arrives, and interrupts T3 as T2 has higher priority than T3, and T2 executes till completion

- In effect, even though T1 has priority than T2, and arrived earlier than T2, T2 delayed execution of T1
- This is “priority inversion” if not acceptable.
- Solution T3 should inherit T1’s priority at step 5

Priority Inheritance Protocol

T1

lock R fails

lock(R)

unlock(R)

T2

lock(R)

T2 acquires

unlock(R)

T3

lock(R)

T3 blocks T2

T3 has priority of T1

T3 deeply blocks T1