SFWR ENG 4AA4

Kemal Ahmed

Fall 2015

Dr. Down

Note: information from the pre-requisite, [SFWR ENG 3DX4](https://drive.google.com/open?id=0BxW61uJyyN8TUjN2X0dwbVBkTVk) will not be included in this summary (although corrections will be).

Contents

[Real-Time Systems 1](#_Toc431408765)

[Classifications 1](#_Toc431408766)

[Task optimization 2](#_Toc431408767)

[Types of Scheduling 2](#_Toc431408768)

[Static 2](#_Toc431408769)

[FIFO 2](#_Toc431408770)

[Period Attributes 3](#_Toc431408771)

[Dynamic 3](#_Toc431408772)

[Multiprocessor 3](#_Toc431408773)

# Real-Time Systems

## Classifications

What happens upon failure to meet deadlines:

* **Soft**: performance is degraded but not destroyed
* **Firm**: a few times will simply degrade performance, but after may lead to system failure
* **Hard**: complete and catastrophic system failure
  + **Safety Critical**: may cause injury / death (a type of hard)

**Forward difference method**: derivatives using 

**Backwards Difference method**: derivatives using 

**Controller** [C(s)]:

**Input** [E(s)]:

**Output** [U(s)]:



# Task optimization

**Task** [T]: 

**Period** [p]: time between tasks are repeatedly released

**Release time** [r]: time it takes to release task

**Execution time** [e]: slowest time task could take to be completed (but assume the tasks will take this long no matter what)

**Deadline** [d]: when task needs to be completed

**Number of tasks** [n]:

**Processor Utilization** [U]: used as a priority level

If ri = 0 and pi = di, then write *Ti* = (*pi* , *ei* )

# Types of Scheduling

## Static

**Static Scheduling**:

* task’s priority is assigned before execution and does not change

### FIFO

**First In First Out (FIFO)**:

* Could cause problems for tasks whose execution time is significantly shorter than the rest when there are deadlines
  + E.g. T1 = (100, 3); T2 = (2, 1)
* A.K.A. **First Come, First Served (FCFS)**

**Schedule**: the order in which tasks will be executed

**Hyperperiod** [H]: the entire length of a cycle, least common multiple

**Frame Size** [f]:

* The best way for computers to segment the schedule in a way that it verify that the appropriate tasks have been executed
* Constraints:

1. 
2. H % f = 0
3. 2f − gcd(pi , f ) ≤ di

**Least Compute Time (LCT)**: tasks with smallest execution times executed first

* Think *greedy*
* Works poorly; worse than RR

**Rate Monotonic (RM)**: shorter period, higher priority

* Think: tasks requiring frequent attention should have higher priority
* *Static scheduling* can be guaranteed to be *feasible* using Rate Monotonic scheduling if: 
  + If the equation fails, it still might be, so draw the whole thing to be safe.

### Period Attributes

**Harmonic**: every task period evenly divides every longer period

* Always feasible with RM schedule

## Dynamic

**Preempting**: splitting tasks up into mini tasks

The only two optimal dynamic priorities are:

* **Earliest Deadline First (EDF)**:
  + more flexible, better U
  + If deadlines < periods, still optimal, but determining feasibility is NP-hard
* **Least Slack Theorem (LST)**: not as popular as EDF

## Multiprocessor

Once you have multiple processors, neither EDF nor RM are guaranteed to work.

Look into first-fit algorithms

**Priority Inversion**: