

CSC 990: Real-Time Systems

Introduction

Steve Goddard
goddard@cse.unl.edu

<http://www.cse.unl.edu/~goddard/Courses/RealTimeSystems>

Jim Anderson

Real-Time Systems

Introduction - 1

Real-time System

- ◆ A **real-time system** is a system whose specification includes both logical and temporal correctness requirements.
 - » **Logical Correctness:** Produces correct outputs.
 - Can be checked, for example, by Hoare logic.
 - » **Temporal Correctness:** Produces outputs at the right time.
 - In this course, we spend much time on techniques for checking temporal correctness.
 - The question of how to specify temporal requirements, though enormously important, is shortchanged in this course.

Jim Anderson

Real-Time Systems

Introduction - 2

Characteristics of Real-Time Systems

- ◆ Event-driven, reactive.
- ◆ High cost of failure.
- ◆ Concurrency/multiprogramming.
- ◆ Stand-alone/continuous operation.
- ◆ Reliability/fault-tolerance requirements.
- ◆ **Predictable behavior.**

Misconceptions about Real-Time Systems

(Stankovic '88)

- ◆ There is no science in real-time-system design.
 - We shall see...
- ◆ Advances in supercomputing hardware will take care of real-time requirements.
 - The old "buy a faster processor" argument...
- ◆ Real-time computing is equivalent to fast computing.
 - Only to ad agencies. To us, it means PREDICTABLE computing.

Misconceptions (Continued)

- ◆ Real-time programming is assembly coding, ...
 - We would like to automate (as much as possible) real-time system design, instead of relying on clever hand-crafted code.
- ◆ “Real time” is performance engineering.
 - In real-time computing, timeliness is almost always more important than raw performance ...
- ◆ “Real-time problems” have all been solved in other areas of CS or operations research.
 - OR people typically use stochastic queuing models or one-shot scheduling models to reason about systems.
 - CS people are usually interested in optimizing average-case performance.

Misconceptions (Continued)

- ◆ It is not meaningful to talk about guaranteeing real-time performance when things can fail.
 - Though things may fail, we certainly don't want the operating system to be the weakest link!
- ◆ Real-time systems function in a static environment.
 - Note true. We consider systems in which the operating mode may change dynamically.

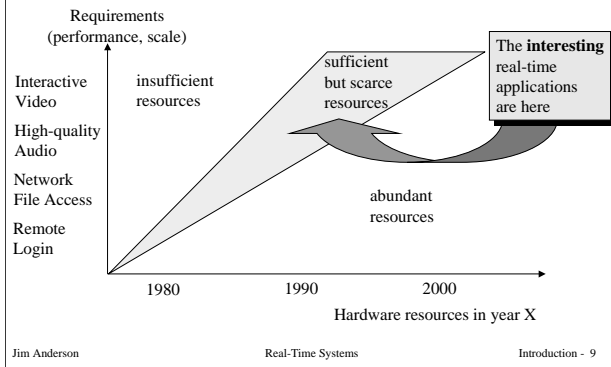
Are All Systems Real-Time Systems?

- ◆ **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.

The "Window of Scarcity"

- ◆ **Resources** may be categorized as:
 - » **Abundant:** Virtually any system design methodology can be used to realize the timing requirements of the application.
 - » **Insufficient:** The application is ahead of the technology curve; no design methodology can be used to realize the timing requirements of the application.
 - » **Sufficient but scarce:** It is possible to realize the timing requirements of the application, but careful resource allocation is required.

Example: Interactive/Multimedia Applications

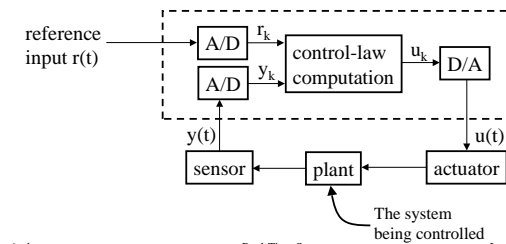


Example Real-Time Applications

(Chapter 1 of Liu)

Many real-time systems are **control systems**.

Example 1: A simple one-sensor, one-actuator control system.



Simple Control System (Continued)

Pseudo-code for this system:

```
set timer to interrupt periodically with period  $T$ ;  
at each timer interrupt do  
  do analog-to-digital conversion to get  $y$ ;  
  compute control output  $u$ ;  
  output  $u$  and do digital-to-analog conversion;  
od
```

T is called the **sampling period**. T is a key design choice. Typical range for T : seconds to milliseconds.

Multi-rate Control Systems

More complicated control systems have multiple sensors and actuators and must support control loops of different rates.

Example 2: Helicopter flight controller.

Do the following in each 1/180-sec. cycle:

validate sensor data and select data source;
if failure, reconfigure the system

Every sixth cycle do:

keyboard input and mode selection;
data normalization and coordinate transformation;
tracking reference update
control laws of the outer pitch-control loop;
control laws of the outer roll-control loop;
control laws of the outer yaw- and collective-control loop

Every other cycle do:

control laws of the inner pitch-control loop;
control laws of the inner roll- and collective-control loop

Compute the control laws of the inner yaw-control loop;

Output commands;

Carry out built-in test;

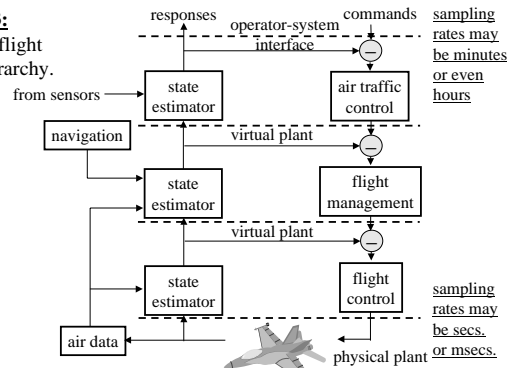
Wait until beginning of the next cycle

Note: Having only **harmonic** rates simplifies the system.

Hierarchical Control Systems

Example 3:

Air traffic-flight control hierarchy.



Jim Anderson

Real-Time Systems

Introduction - 13

Signal-Processing Systems

◆ **Signal-processing systems** transform data from one form to another.

◆ Examples:

- » Digital filtering.
- » Video and voice compression/decompression.
- » Radar signal processing.

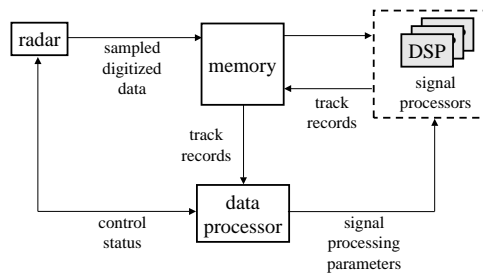
◆ Response times range from a few milliseconds to a few seconds.

Jim Anderson

Real-Time Systems

Introduction - 14

Example: Radar System



Other Real-Time Applications

◆ Real-time databases.

- Transactions must complete by deadlines.
- **Main dilemma:** Transaction scheduling algorithms and real-time scheduling algorithms often have conflicting goals.
- Data may be subject to **absolute** and **relative temporal consistency** requirements.

◆ Multimedia.

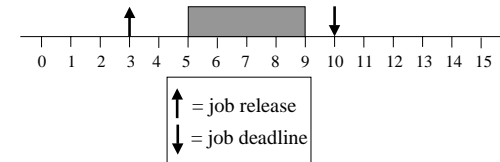
- Want to process audio and video frames at steady rates.
 - TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
 - Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec.
- **Other requirements:** Lip synchronization, low jitter, low end-to-end response times (if interactive).

Hard vs. Soft Real Time

(Chapter 2 of Liu)

- » **Task:** A sequential piece of code.
- » **Job:** Instance of a task.
- » Jobs require **resources** to execute.
 - **Example resources:** CPU, network, disk, critical section.
 - We will simply call all hardware resources “processors”.
- » **Release time of a job:** The time instant the job becomes ready to execute.
- » **Deadline of a job:** The time instant by which the job must complete execution.
- » **Relative deadline of a job:** “Deadline – Release time”.
- » **Response time of a job:** “Completion time – Release time”.

Example



Job is released at time 3.
It's (absolute) deadline is at time 10.
It's relative deadline is 7.
It's response time is 6.

Hard Real-Time Systems

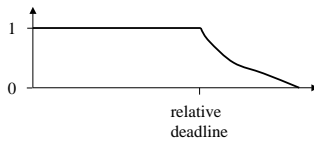
- ◆ A **hard deadline** *must* be met.
 - » If *any* hard deadline is *ever* missed, then the system is **incorrect**.
 - » Requires a means for **validating** that deadlines are met.
- ◆ **Hard real-time system:** A real-time system in which all deadlines are hard.
 - » We mostly consider hard real-time systems in this course.
- ◆ **Examples:** Nuclear power plant control, flight control.

Soft Real-Time Systems

- ◆ A **soft deadline** may *occasionally* be missed.
 - » **Question:** How to define “occasionally”?
- ◆ **Soft real-time system:** A real-time system in which some deadlines are soft.
- ◆ **Examples:** Telephone switches, multimedia applications.

Defining “Occasionally”

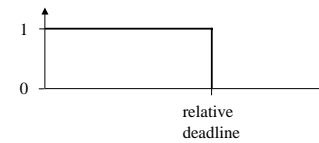
- ◆ **One Approach:** Use probabilistic requirements.
 - » For example, 99% of deadlines will be met.
- ◆ **Another Approach:** Define a “usefulness” function for each job:



- ◆ **Note:** Validation is trickier here.

Firm Deadlines

- ◆ **Firm deadline:** A soft deadline such that the corresponding job’s usefulness function goes to 0 as soon as the deadline is reached (late jobs are of no use).



- ◆ Firm deadlines are not considered in Liu’s book.

Reference Model

(Chapter 3 of Liu)

- ◆ Each job J_i is characterized by its release time r_i , absolute deadline d_i , relative deadline D_i , and execution time e_i .
- ◆ Sometimes a range of release times is specified: $[r_i^-, r_i^+]$. This range is called **release-time jitter**.
- ◆ Likewise, sometimes instead of e_i , execution time is specified to range over $[e_i^-, e_i^+]$.
 - » **Note:** It can be difficult to get a precise estimate of e_i (more on this later).

Periodic, Sporadic, Aperiodic Tasks

(Or, let the terminology wars begin...)

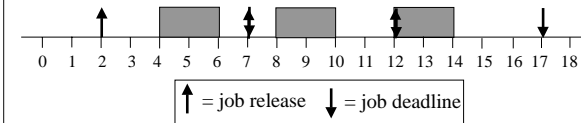
- ◆ **Periodic task:**
 - » We associate a **period** p_i with each task T_i .
 - » p_i is the minimum time between job releases.
- ◆ **Sporadic and aperiodic tasks:** Released at arbitrary times.
 - » **Sporadic:** Has a hard deadline.
 - » **Aperiodic:** Has no deadline or a soft deadline.

Warning!

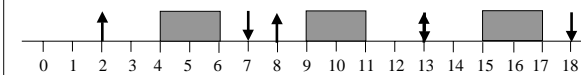
- ◆ What Liu calls “periodic”, the rest of the world calls “sporadic”.
- ◆ In the rest of the world, the period p_i of a periodic task T_i gives the **exact** spacing between job releases.

Examples

A periodic task T_i with $r_i = 2$, $p_i = 5$, $e_i = 2$, $D_i = 5$ executes like this according to the rest of the world:



According to Liu, it could execute like this:



To the rest of the world, this is a sporadic task.

Some Definitions for Periodic Task Sys.

- ◆ The jobs of task T_i are denoted $J_{i,1}, J_{i,2}, \dots$.
- ◆ $r_{i,1}$ (the release time of $J_{i,1}$) is called the **phase** of T_i .
 - » **Synchronous System:** Each task has a phase of 0.
 - » **Asynchronous System:** Phases are arbitrary.
- ◆ **Hyperperiod:** Least common multiple of $\{p_i\}$.
- ◆ **Task utilization:** $u_i = e_i/p_i$.
- ◆ **System utilization:** $U = \sum_{i=1..n} u_i$.

Task Dependencies

- ◆ Two main kinds of dependencies:
 - » Critical Sections.
 - » Precedence Constraints.
 - For example, job J_i may be constrained to be released only *after* job J_k completes.
- ◆ Tasks with no dependencies are called **independent**.
 - » In the first half of the course, we will consider only independent tasks.

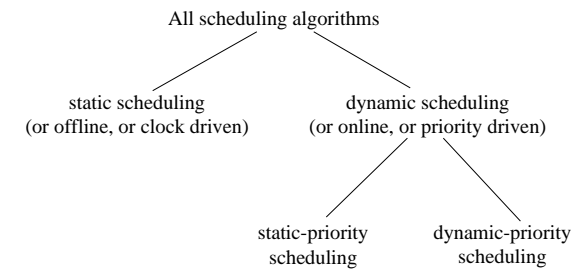
Scheduling Algorithms

◆ We are generally interested in two kinds of algorithms:

- 1 A **scheduler** or **scheduling algorithm**, which generates a schedule at runtime.
- 2 A **feasibility analysis algorithm**, which checks if timing constraints are met.

< Usually (but not always) Algorithm 1 is pretty straightforward, while Algorithm 2 is more complex.

Classification of Scheduling Algorithms



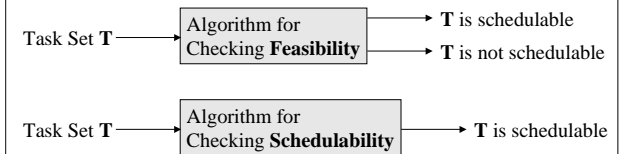
Optimality and Feasibility

- ◆ A schedule is **feasible** if all timing constraints are met.
 - The term “correct” is probably better — see the next slide.
- ◆ A task set T is **schedulable** using scheduling algorithm A if A always produces a feasible schedule for T.
- ◆ A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm).
 - Can similarly define optimality for a class of schedulers, e.g., “an optimal static-priority scheduling algorithm.”

Feasibility versus Schedulability

To most people in real-time community, the term “**feasibility**” is used to refer to an **exact** schedulability test, while the term “**schedulability**” is used to refer to a **sufficient** schedulability test.

You may find that these terms are used somewhat inconsistently in the papers we read.



Real-Time Research Repository

- ◆ For information on real-time research groups, conferences, journals, books, products, etc., have a look at:

» <http://cs-www.bu.edu/pub/ieee-rts/Home.html>