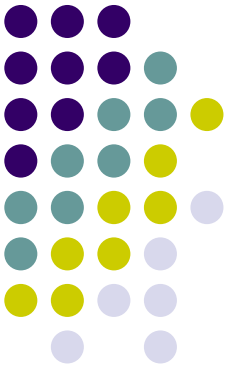
Introduction to Real-Time Systems

Ingo Sander
ingo@kth.se



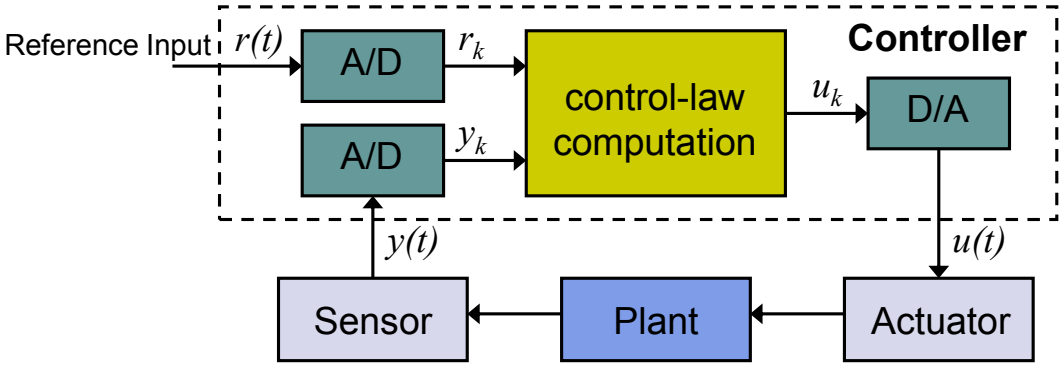
Based on Liu: Chapter 1

Real-time

- A system is classified as *real-time* if it is required to complete the work on a timely basis
  - It does not mean that a system must meet some timing deadlines!
- It is of utmost importance that the correctness of the timely behavior of a real-time system can be validated
  - often real-time systems are also safety-critical, a missing of a deadline can lead to disastrous consequences
  - bugs in real-time systems may be very costly to fix afterwards, especially in embedded systems

Example for real-time system Digital process control

- The digital controller regulates a controlled system "plant" (can be engine, patient, …) by comparing the measured state y_k with the desired state r_k and calculating output results u_k to stimulate the "plant"



Digital process controller

- Pseudo-Code:

  set timer to interrupt periodically with period T
  at each timer interrupt do
  analog-to-digital conversion to get y
  compute control output u
  output u and digital-to-analog conversion of u
  end do

Digital process controller

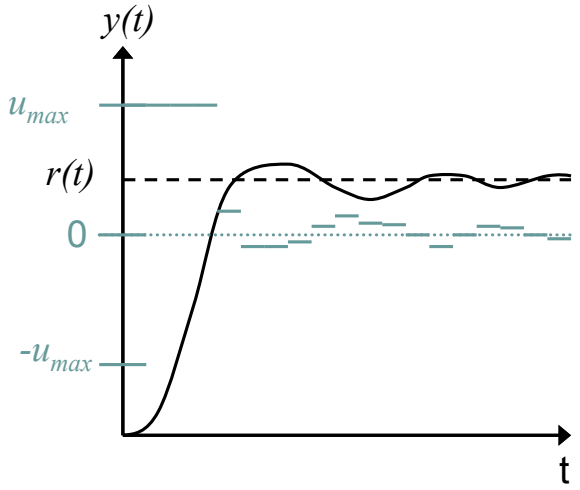
- Efficient control of the plant depends on
  - Correct control law computation
  - Correct reference input
  - Accuracy of sensor measurements
    - resolution of sampled data (number of bits per sample)
    - timing of clock interrupts (samples per second)

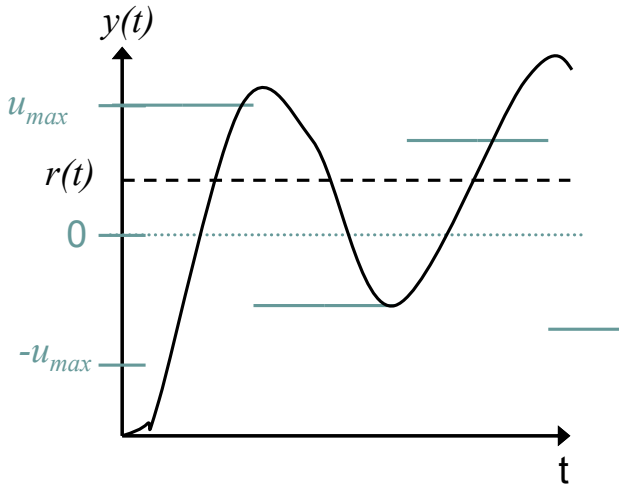
Digital process controller

- The time T between two consecutive measurements is called the sample period
  - small T better approximates the analogue behavior
  - large T means less processing power is needed
  - Compromise is needed!
- If T is too large oscillation will result as system tries to adapt

Digital process controller

- Short sampling period
  - Controller reaches $r(t)$
- Long sampling period
  - oscillation





Digital process controller

- Rule of thumb for signal processing applications:
  - Sampling period should be chosen in such a way that the *rise time* R is 10 to 20 times larger than the sample period T
  - *Rise Time* is usually defined as the amount of time that it takes the plant to go from 10% to 90% of the height of the step

Digital process controller

- Theoretical lower limit is given by *Nyquist sampling theorem*:
  - Any time-continues signal of bandwidth B can be reproduced faithfully from its sampled values, if and only if the sampling rate is $2B$ or higher
  - Example CD
    - Bandwidth: 22050 Hz (given by human ear)
    - Sampling Frequency: 44100 Hz

Digital process controller

- Systems often consist of multiple sensors and actuators
- Sensors and actuators need different sampling periods
- Such systems are called *multi-rate systems*
- In such systems periods are usually related in a *harmonic way*, i.e. each longer period is an integer multiple of the shortest period

Digital process controller

- So far we have made the following assumptions
  1. sensor data give accurate estimates of the state variables that are monitored and controlled
  2. sensor data give state of the plant (sometimes values are measured indirectly)
  3. all parameters representing the dynamics of the plant are known

Digital process controller

- If the previous assumptions are not fulfilled the control law computation must
  - try to make a correct estimation of the state of the plant based on “noisy” sensor values
  - compensate for the less accurate measurement by an improved model and more complex computation

Other examples

- Other examples for real-time systems
  - Automotive applications
    - ABS brake system in a car (distributed)
    - Airbag controller
  - Multimedia applications
    - Voice
    - Video (requires much more processing power)
  - Real-time data bases

Types of real-time applications

1. purely cyclic
   - every task executes periodically
   - demands do not vary from period to period
   - example: simple digital controllers
2. mostly cyclic
   - some asynchronous external events
   - example: more complex process controllers

Types of real-time applications

3. asynchronous and somewhat predictable
   - duration between consecutive executions of a task vary
   - statistical distribution of execution pattern known
   - example: bursty multimedia systems
4. asynchronous and unpredictable
   - reaction to asynchronous events with tasks that have high run-complexity

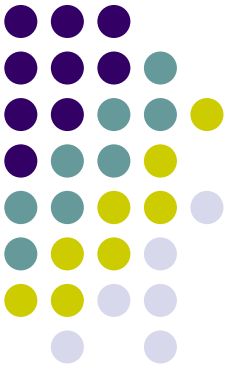
Designing real-time systems

- It is much easier to reason about synchronous, cyclic and predictable systems
- Safety and correctness are first class citizens
  - take a conservative design approach
  - efficiency is important, but correctness is number one

A Reference Model of Real-Time Systems

Ingo Sander

ingo@kth.se



Based on Liu: Chapter 2 and 3

Modeling Real-Time Systems

- A good model
  - abstracts from irrelevant details
  - provides the formalism that allows to reason about properties of the system
  - is independent of the implementation language
  - comes with a consistent terminology

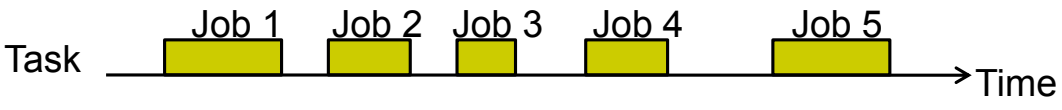
Reference Model of Real-Time Systems

- The reference model used in Liu is characterized by
  1. *workload model* that describes the applications in the system
  2. *resource model* that describes the system resources available to the applications
  3. *algorithms* that define how the application system uses the resources at all times

This lecture focuses on (1) and (2), the algorithms are discussed in later lectures

Jobs and Tasks

- A *job* is a unit of work that is scheduled by the system
  - computation of a control law (digital controller)
  - FFT computation on sample data
  - transmission of a packet
- A *task* is a set of related jobs which jointly provide a system function
  - the set of jobs that form a task like “maintain speed” in an automatic cruise control
    - includes jobs like “monitor cruise speed”, “adjust engine revolution”



Processors and Resources

- A *job* executes on a *processor* and may depend on some *resources*
  - A *processor P* is an active object
    - CPU, Transmission Link, Server
    - Processors have attributes (preemptivity, context switch time, speed)
    - Two processors are of the same type, if they are functionally identical and can be used interchangeably
  - A *resource* is a passive object
    - memory, semaphore
    - resources are in this model reusable and are not consumed after usage

Resources

- Each resource may have one or more units, and each unit is used in a mutually exclusive manner
- A job must obtain a unit of a needed resource and release it after usage
- A resource is *plentiful*, if no job is ever prevented from execution by the lack of this resource
  - Such resources are usually not part of the model, since they do not affect the behavior

Jobs

- According to the model the basic component of any real-time application are jobs
- The operating system treats jobs as a unit of work and allocates processors and resources

Job parameters

- A job J_i is characterized by many parameters:
  - execution time
  - deadlines
  - preemptivity
  - resource requirements
  - soft or hard real time

Execution time

- e_i is the amount of time required to complete J_i, when it executes alone and has all resources it needs
- The execution time depends on speed of processor and complexity of job, but may vary due to
  - caches, pipeline, conditional branches
  - often intervals are given [e_i^- , e_i^+]
- Especially for hard-real time systems e_i often denotes the maximum execution time and is used for calculation
  - can lead to worse performance, but gives a safe bound

Release time

- The *release time* r_i of a job is the instant of time at which the job becomes available for execution
- Release time may be jittery so that r_i is in the interval [r_i^- , r_i^+]
- Jobs can be scheduled and executed at any time after its release time, whenever its data and control dependency conditions are met

Response time

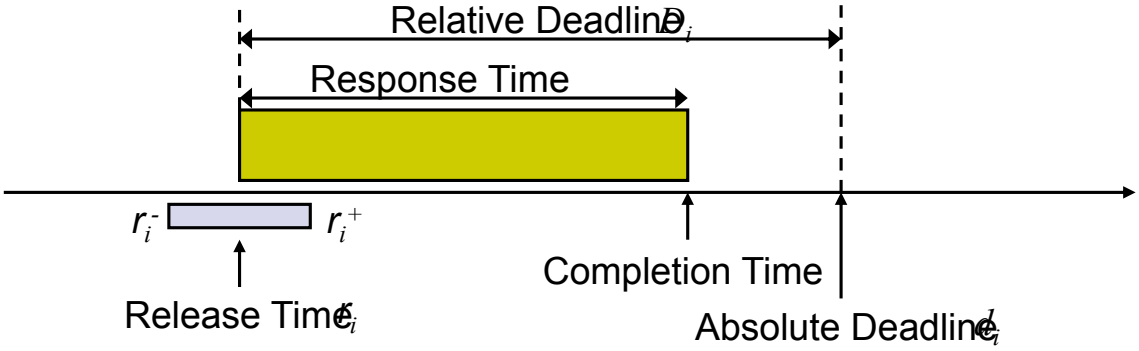
- The *response time* is the length of time from the release time of the job to the time it completes

Deadlines

- The *deadline* of a job is the instant of time by which its execution is required to be completed
  - *Relative deadline* D_i: the maximum allowable response time of a job
  - *Absolute deadline* d_i: the time at which a job is required to be completed (often simply called deadline)
    - release time + relative deadline

Feasible interval

- The *feasible interval* is the time interval $(r_i, d_i]$ between release time and absolute deadline



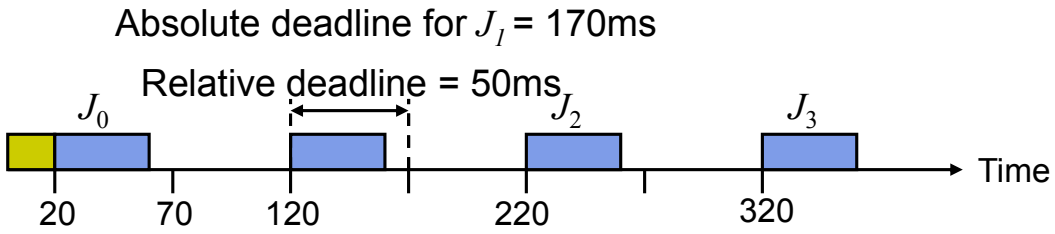
The notation $(r_i, d_i]$ means the interval that begins immediately after r_i and ends at d_i.

Example: Heating furnaces

- System monitors and controls several heating furnaces (ovens)
- System takes 20ms to initialize
- After initialization, every 100ms, the system
  - samples and reads each temperature sensor
  - computes the control law of each furnace to determine flow rates of fuel, air and coolant
  - adjust flow rates

Example: Heating furnaces

- Computations are periodic and can be stated in terms of release times of the jobs computing the control law: $J_0, J_1, ..., J_k, \ldots,$ where the release time of $J_k = (20 + (100k))$ ms
- Suppose job must be finished at $d_k = (70 + (100k))$ ms



Tardiness and Lateness

- The tardiness and lateness of a job measures how late a job completes respective to its deadline
  - lateness = completion time – absolute deadline
    - lateness can be positive or negative
  - tardiness = maximum (0, lateness)
    - can only be positive

Tardiness and lateness are useful measures for soft real-time systems!

Hard and soft deadlines

- A deadline or timing constraint is considered *hard,* if the failure to meet it is considered to be a fatal fault
- A deadline or timing constraint is considered *soft,* if the failure to meet it undesirable, but does not lead to fatal faults
- Definition is not clear cut, since term fatal can be interpreted in different ways!

Hard and soft real-time systems

- A hard real-time system is a system containing mostly hard real-time tasks
  - automatically controlled train, airbag controller
- A soft real-time system is a system containing only soft real-time tasks
  - multimedia applications
- Border between hard and soft-real time is not fixed!

Workload parameters

- The workload on a processor consists of the jobs that shall be executed on the processor
  - A set of related jobs that execute a system function is a task
- Execution and resource parameters are obtained from measurements and analysis
- These parameters must be known in order to give guarantees on deadlines!

Jobs in a task

- Jobs in a task may be related to each other, which gives additional constraints
  - jobs must execute in certain order (precedence constraints)
  - jobs must complete within a certain amount of time (temporal distance constraints)
  - jobs may have data dependencies with each other

Periodic task model

- A *periodic task* T_i is a sequence of jobs $J_{i,1}$, $J_{i,2}$, ..., $J_{i,n}$ that can be executed repeatedly
  - the *period* p_i is the minimum length of all time intervals between release times of consecutive jobs
  - the *execution time* e_i of T_i is the maximum execution time of all jobs in the task
  - the *phase* Φ_i is the release time of the first job in T_i

Notation Periodic Task

- A periodic task is described by phase Φ_i, period p_i, execution time e_i, and relative deadline D_i
- Usually a 4-tuple (Φ_i, p_i, e_i, D_i) is used
- If not given, default values for phase (Φ_i = 0) and or deadline (p_i = D_i) are used

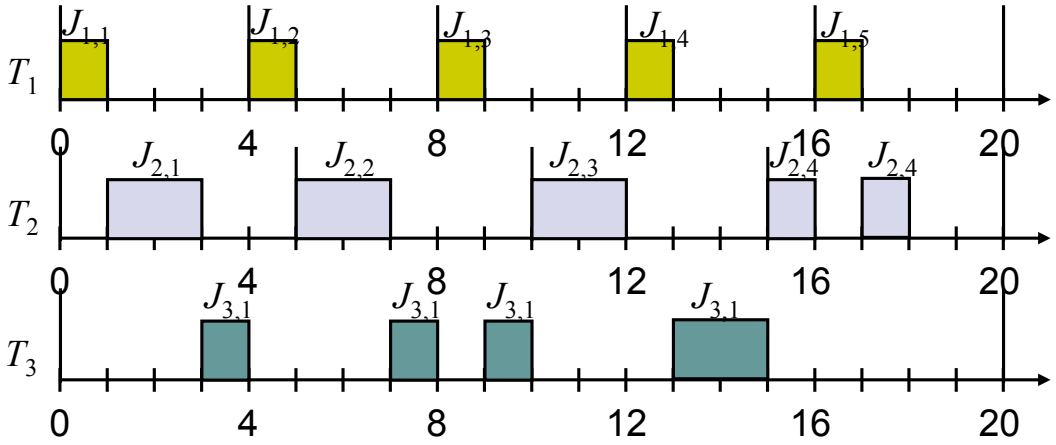
Notation Periodic Task

- Examples:
  - T_1 = (1, 10, 3, 6) => Φ_i = 1, p_i = 10, e_i = 3, D_i = 6
    - $J_{1,1}$ is released at $r_{1,1}$ = 1, deadline is at d_i = 7
    - $J_{1,2}$ is released at $r_{1,2}$ = 11, deadline is at d_i = 17
  - T_1 = (10, 3, 6) => Φ_i = 0, p_i = 10, e_i = 3, D_i = 6
  - T_1 = (10, 3) => Φ_i = 0, p_i = 10, e_i = 3, D_i = 10

Periodic task model

- The *utilization* u_i of a task T_i is the ratio e_i / p_i
- The *total utilization* U of all tasks in the system is the sum of the utilizations of all individual tasks in it
- The *hyper-period* H of a set of periodic tasks is the least common multiple of their periods

Periodic tasks



- $T_1 = (4,1);\ T_2 = (5,2);\ T_3 = (20,5)$
- Utilization: $u_1 = 0.25;\ u_2 = 0.4;\ u_3 = 0.25 \Rightarrow U = 0.9$

Periodic task model

- Assumption
  - We will often assume that for every task a job is released and becomes ready at the beginning of each period and must complete by the end of the period
- Model is widely used, since
  - many applications fit into this model
  - model allows formal reasoning

Aperiodic and sporadic tasks

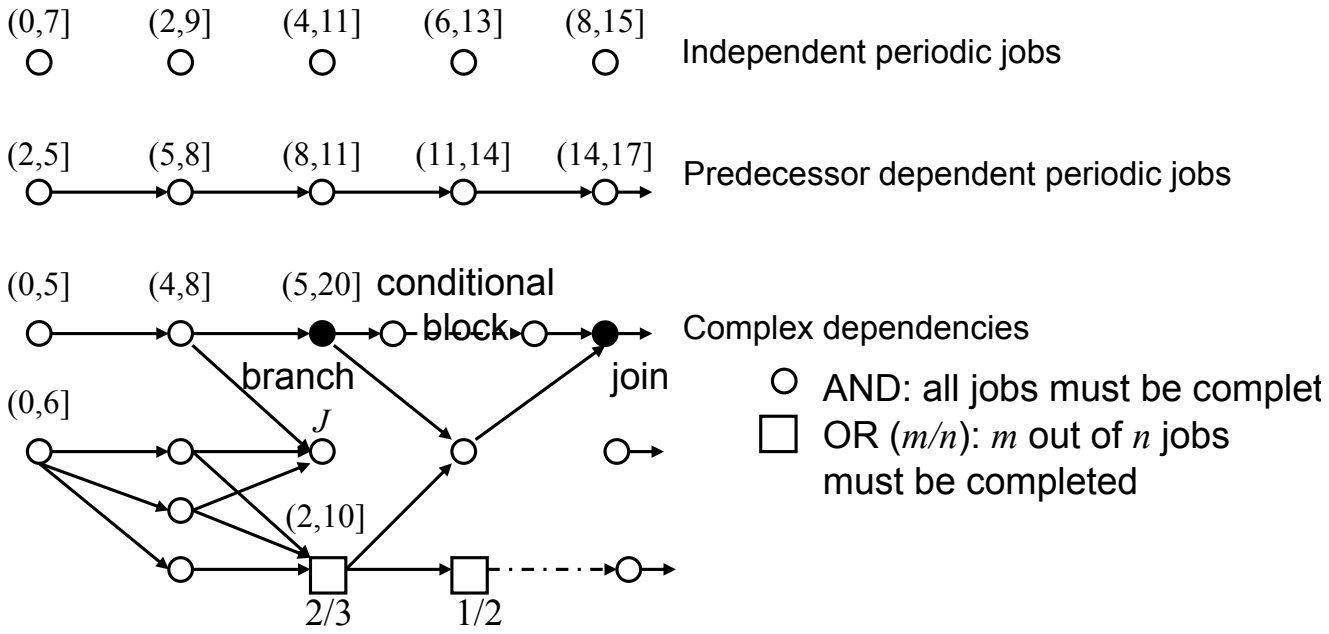
- The processor executes an *aperiodic* or *sporadic job* when it responds to external events that come at irregular intervals
  - Aperiodic jobs have soft deadlines
  - Sporadic jobs have hard deadlines
- An aperiodic or sporadic task is a stream of aperiodic or sporadic jobs, respectively
- Aperiodic and sporadic jobs occur in real-time systems, and aggravate the analysis of real-time systems substantially!

Precedence constraints

- The jobs in a task may be constrained to execute in a particular order (precedence constraints)
  - J_i is a *predecessor* ($J_i < J_k$) of another job J_k, if J_k cannot begin execution until the execution of J_i completes
  - J_i is an *immediate predecessor* of another job J_k, if $J_i < J_k$ and there is no other job J_j such that $J_i < J_j < J_k$
  - J_i and J_k are *independent*, when neither $J_i < J_k$ nor $J_k < J_i$
- A job with predecessors becomes ready for execution when the time is at or after its release time and all of its predecessors are completed

Task Graphs

- A task is an extended precedence graph



Preemptivity of jobs

- The interruption of a job is called *preemption*
- A job is *preemptable* if its execution can be suspended at any time to allow the execution of other jobs, and later can be resumed from the point of suspension
  - requires reentrant functions (check IL2206 or Labrosse)
- A job is *non-preemptable*, if it must be executed from start to completion without interruption

Preemptivity of jobs

- The (non-)ability to preempt a job affects the scheduling algorithm
- The time it takes to switch between two jobs is the *context switch time* (overhead for scheduling)

Scheduling

- Jobs are scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols
- A scheduler assigns jobs to processors
- A schedule is an assignment of all the jobs in the system on the available processors

Valid schedule

- A *valid schedule* satisfies the following requirements
  1. every processor is assigned to at most one job at any time
  2. every job is assigned to at most one processor at any time
  3. No job is scheduled before its release time
  4. Depending on the scheduling algorithm(s) used, the total amount of processor time assigned to every job is equal to its maximum or actual execution time
  5. All the precedence and resource usage constraints are satisfied

Feasibility, optimality and performance measures

- A *feasible schedule* is a valid schedule, if every job meets its timing constraints
- A hard real-time algorithm is *optimal*, if it always produces a feasible schedule if the given set of jobs has a feasible schedule
- For soft-real time systems performance can be described by
  - *miss rate*: percentage of jobs that are executed but completed too late
  - *loss rate:* percentage of jobs that are discarded and not executed at all

Summary

- Presentation of reference model
- Timing constraints
- Soft and hard real-time
- Precedence constraints
- Schedules

More information in chapter 2 and 3 of Liu's book…