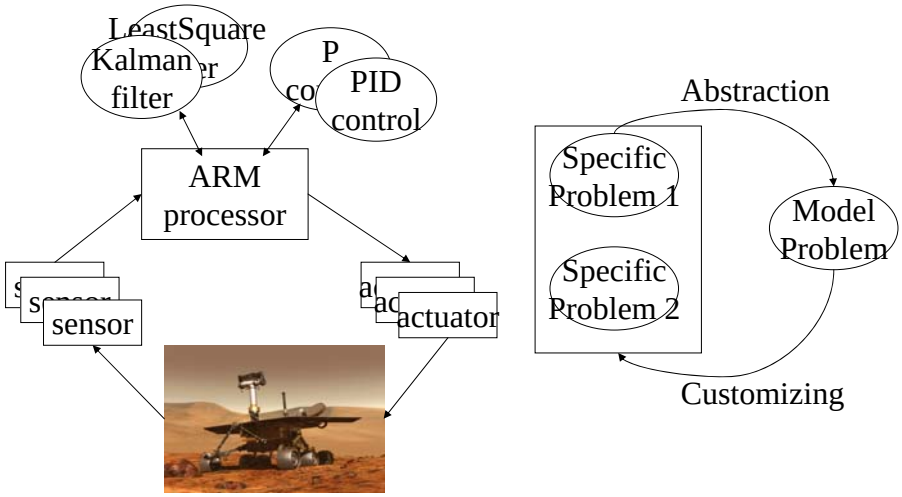
Real-Time Task Model
- Chapter 3 -

"Monolithic approach" Good enough?

- No!
  - For a complex system, it is really hard to design a single superloop
  - Really hard to validate cross-related temporal requirements
- So, we need a more structured way to look at the problem
  - Reference model: abstracted view (look at only core) of
    - Workload
    - Resource
  - Easy to form a structured way of design and validation
  - Can be generalized for many variations of problems

Why Reference Model?



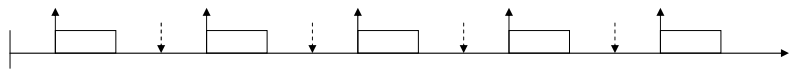
The Objective of A Reference Model

- Directly handling wide variety of details (Kalman or Sorting, C++ or Ada, Pentium III or ARM-7, Unix or Windows)?
  - have to deal with irrelevant details
  - hard to be generalized
- A Reference Model captures only relevant characteristics and categorize the problem space
  - workload model
  - resource model
  - algorithms
- Like doctor's reference handbook, a reference model help you
  - classify a given real-time problem
  - know if it is a solved problem or open problem
  - know if it is a easy problem or hard problem

Overview

- Workload model (characterization of applications)
  - Jobs and tasks
  - Release time (periodic/aperiodic/sporadic)
  - Deadline (absolute/relative, hard/soft, single stage/end to end)
  - Execution time (deterministic/stochastic)
  - Temporal distance and precedence constraints
- Resource model
  - Processors and resources
  - Utilization of resource
- Scheduling algorithms
  - Priority/preemptability/blocking
  - Scheduler and schedule

Workload Model

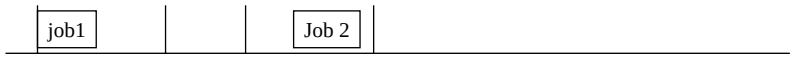


0 (system startup time)

- Task and job
- Task offset
- Release time of jobs (periodic, sporadic, aperiodic)
- Deadline of jobs (hard or soft)
- Execution time of jobs (deterministic or stochastic)

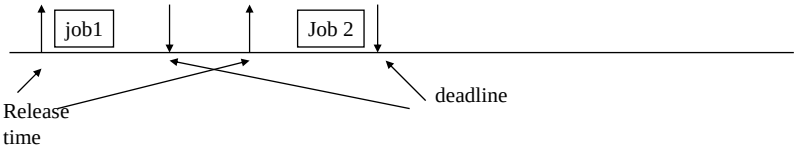
Jobs and Tasks

- 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.



Release Time and Deadline

- Release time is the instant at which the job becomes ready to execute
- Deadline is the time by which the job should complete.



Periodic, Aperiodic and Jitter in Release times

- A task is ***periodic***, if the release times are periodic, e.g., every second.
- But realistically, we cannot do it exactly every second in a physical system. There could be some small errors such as a window of " $\pm$ 1 msec". This is known as ***jitter***
- If the release time is irregular. It is known as ***aperiodic***. If the aperiodic arrivals have a minimal distance that separates two consecutive release times, it is also known as ***sporadic*** task. (This term is not universally agreed.)

Absolute and Relative Deadlines

- The common form of deadlines are ***absolute deadlines*** where deadlines are specified in, well, absolute times. Train and airlines schedules are absolute but soft deadlines.
- Normally, ***relative deadlines*** are related to the release time. For example, 10 msec after the release time.
- Sometimes, it is specified to be *n* units of time after an event is detected. For example, upon the detection of a frontal collision, the airbag must be deployed within *k* msec. In this case, the occurrence of the event is the release time.
- ***Laxity*** is a measure of how long is left to hit the deadline, say, 10 sec left to launch the Shuttle.

Hard and Soft Deadlines

- Deadlines are considered to be ***hard*** if failing to meet them is considered as an application failure leading to catastrophic consequences on the controlled environment.
- A deadline is considered to be ***soft***, if it is specified probabilistically. For example, a radar system's ability to detect and display a plane by some given time is usually specified in the following style:
  - When a plane is over the horizon, it must be detected and displayed on operator's screen no later than 1 second 99.95 % of the times.
- A deadline is also considered to be ***soft*** if a late completion gracefully degrades the performance of the system without causing damage to the controlled environment

Deterministic and Stochastic Execution Times

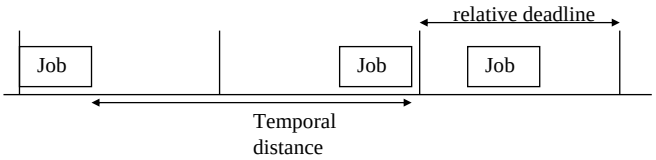
- Literally speaking, there is no such thing as "deterministic execution times". There will always be some variations.
- Deterministic execution time in practice means that we know the worst-case execution time. In practice, it often implies that the worst-case execution time is not too far away from average execution time so that it is practical to use it. Most numerical computation used in control fits this model.
- Stochastic execution time in practice means that the use of the worst-case execution time is no longer practical since it is too far off from the average. Compressed motion video is a good example.

Periodic Task Model

- Periodic tasks are the "work horse" of real-time systems and they play a key role in real-time systems.
- A periodic task, T_i, is characterized by
  - Phase (or offset) ϕ_i : the starting time of the task, i.e., the first release time ($r_{i,1}$). The default is 0.
  - 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 end of period

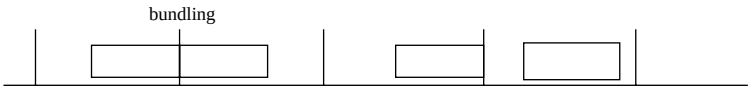
Temporal Distance - 1

- Sequential temporal distance constraint: how far the completion time of two consecutive jobs can be separated.
- Example Application: Advanced phase array radar. Temporal distance and deadline specify the window of time at which another look (SAMPLING) of a target is needed. Temporal distance and deadline are computed online according to the nature and speed of the target.
- Quiz: why not using a periodic model?



Temporal Distance - 2

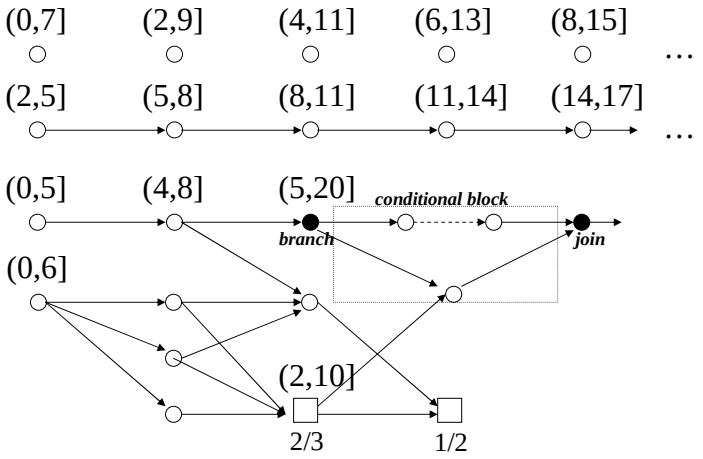
- Periodic model is still in use. But it permits bundling, which wastes antenna resource. If next sample completes at the end of next period, the wide separation will result in more error.
- Instance by instance temporal distance (from the current job's completion time to the completion time of next job) gives more precise control on the use of the antenna.



Precedence Constraints

- Precedence relation says you can't start until your predecessor has done.
- Example: "Data producer job" has a precedence constraint with "Data consumer". For a sensory data fusion, filtering of individual sensor data should be completed before their fusion.

Precedence Constraint Representation (Task Graph)



Resource Model (Processors and Resources)

- Active resources: processor, bus, disks and communication channels where a job executes or transmits over it. The size of the job and the speed of the active resources determine the execution/transmission time. Usually, active resources are scheduled, for example, scheduling the CPU or LAN.
- Passive resources or just resources: shared variables, semaphores, or library routine etc, which the jobs need to make execution (transmission). We do not associate speed with a passive resource.

Utilization of Active Resource by a task

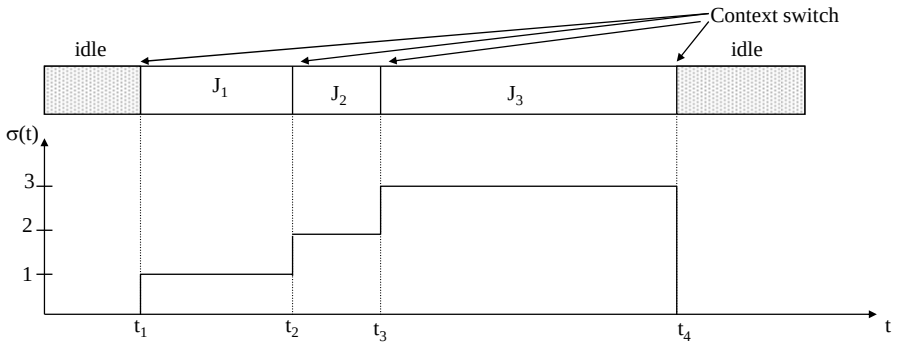
- A periodic task's utilization U_i of an active resource is the ratio between its execution time and period: $U_i = C_i/p_i$
- Given a set of periodic tasks on an active resource, e.g. the CPU, the CPU's utilization is equal to the sum of periodic tasks' utilization:

$$U = \sum_i \frac{C_i}{p_i}$$

- Given a set of aperiodic tasks, the average CPU utilization is the ratio of average job execution time and average job inter-arrival time.

Scheduling Model "Schedule and Scheduler"

- Given a set of jobs, J= {J1,…,Jn}, a *schedule* σ(t) is an assignment of jobs to the resource, so that each task is executed until completion.
- σ(t) is an integer step function and σ(t)=k, with k>0, means that job Jk is executing at time t, while σ(t)=0 means that the CPU is idle.



Schedule and Scheduler

- Scheduler is the module that implements the scheduling algorithm. Each processor is assigned one job at a time.
  - Jobs become ready to execute after their release time and after requested resources are allocated and precedence constraints are satisfied
  - Jobs are executed in the priority order, preemption carries out and processor is de-allocated immediate after a job is done.
- A schedule is said to be *feasible* if all tasks can be completed according to a set of specified constraints.
- A set of tasks is said to be *schedulable* if there exists at least one algorithm that can produce a feasible schedule for the task set.

Preemptive vs Non-preemptive scheduling

- Preemption: A *preemptive* schedule is a schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy
- CPU is preemptable.
- Communication channel is non-preemptable.
- Passive resource, e.g., shared data, is not preemptable – mutually exclusive.

Classification of Scheduling Algorithms

- **Preemptive/ Non-preemptive**. In preemptive scheduling, the running task can be interrupted at any time to assign the processor to another active task. On the other hand, in non-preemptive scheduling, a task, once started, is executed by the processor until completion.
- **Off-line/On-line/Clairvoyant**. The scheduling algorithm runs off-line/on-line. In the off-line case, the generated schedule is stored in a table and later executed by a dispatcher. In on-line case, the schedule is determined online by looking at jobs in the ready queue. Clairvoyant (like GOD!) know even the future jobs and schedule jobs accordingly.