Important issues in the course

- Predictability and timing analysis of algorithms
- Timing aspects on memory management, cache handling, etc.
- Task scheduling (static, dynamic, …)
- Priority (static, dynamic, inheritance, …) and queue handling
- Task partitioning, grouping, allocation and load distribution
- Resource reservation using semaphores, monitors, message passing, ...
- Device (sensor & actuator) sharing and allocation
- Reliability, availability, safety, security and maintainability
- Control loop (feedback) issues
- Timing aspects on interconnection, networking and communication
- Energy aware scheduling (to save battery life) etc.

What is essential

- Timelines (do what is needed in time)
- Efficient utilisation of shared resources
  - efficient distribution (space)
  - efficient scheduling (time)
- To avoid conflicts when sharing resources (in space and time)

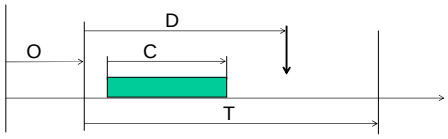
Conflict avoidance

- Time analysis, time planning and synchronisation
- Reservation and locking of resources
- Queue and priority handling
- Negotiation, dialogue (protocol)

Task = unit of work with deadline

Task (instance) parameters = {C, T, D, O}

C = Computation or execution time
T = Time period, for repetition of task
D = Deadline, relative to release time
O = Offset or phase, from first triggering of a periodic task activation

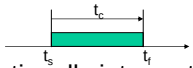


Basic assumptions

- A computer is a resource that can perform some X units of work per second
- A task defines repetitive jobs requiring C units of work to be done within each period T (or deadline D) when released and started until it is finished
- The execution time of a task is however often expressed as a percentage of the processors execution capacity (be aware)
- The ordering of the different activities defined as a task is described by a program

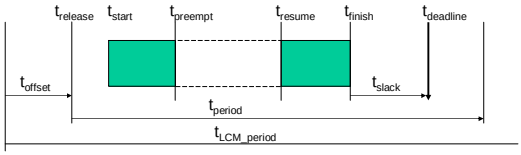
Basic assumptions, cont.

- The job performed by a task has a start time t_s and a worst case finishing time t_f



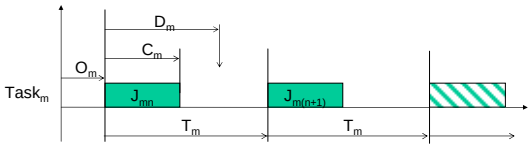
- A job can optionally interrupt itself and wait for a re-activation event
- A job can be preempted by another job with higher priority

Task timing terminology



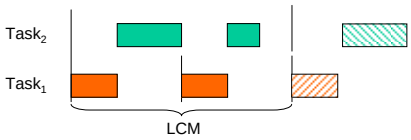
R = release (request or arrival) time (ready to start execution)
S = start time (computing job start to execute)
F = finish time (computing job finish execution)
D = deadline (last acceptable finishing time)
Times are usually given relative to release time
Observe! The deadline is often equal to the period but can be shorter

Task performing periodic jobs



In this case a task is activated periodically to perform periodic jobs

Release, start and finish



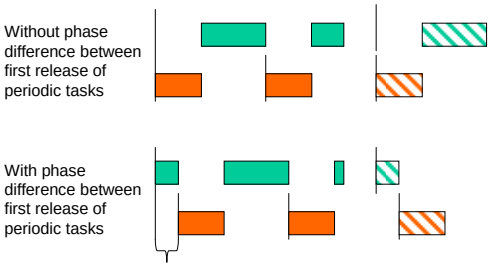
Both tasks are released at the same time and in phase.

The highest priority task start at once.

The other lower priority task has to wait until it can start.

The lower priority task is disturbed twice, first before it starts and secondly during computing and can only finish after that it has resumed its execution after being preempted (interrupted by the scheduler).

Phase (offset) between task periods



Phase or offset between periodic tasks relative to start of LCM period

Analysis and design issues

- Task execution times and processor utilization
- Feasible allocation in time (schedule) and space
- Inference and effects from other tasks
- Communication times and delays
- Dependability (timing is meaningless unless the system also is reliable, available, …)

Utilization

Total processor utilization of a task set:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i}$$

Lower bound of the processor utilization for N tasks:

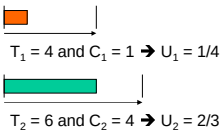
$$U \leq N\left(2^{\frac{1}{N}} - 1\right)$$

TASK SET

This is an example of a simple task set.

It consists of two tasks with different:

- period T
- computation time C
- measured in some time unit
- resulting in a processor utilization for the task set



$T_1 = 4$ and $C_1 = 1$ → $U_1 = 1/4$

$T_2 = 6$ and $C_2 = 4$ → $U_2 = 2/3$

Response time

1. The maximum interference I_i of a task τ_i occurs when all higher priority tasks arrives at the same time as τ_i (i.e., the critical instant of τ_i).
2. This means that the task is disturbed and delayed as often as possible during its period.
3. This thus results in the maximum response time:

   $R_i = C_i + I_i$

   where C_i is the worst case computation time and I_i is the max interference of τ_i

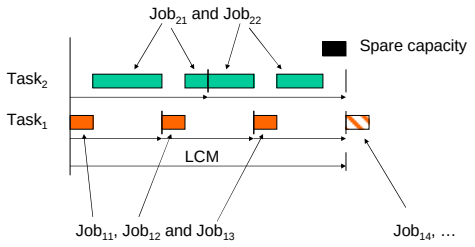
Analysis and design issues, cont.

- Partitioning of one large global task into several smaller and more local tasks
- Allocation and distribution of tasks to processing nodes
- Scheduling (or time ordering) of tasks
- Allocation and scheduling of communication resources

Analysis and design issues, cont.

- To what degree do we utilize the capacity of the processor?
- Can tasks be allocated so that no processor is overloaded ?
- Is it feasible to design a schedule?
- What is the least common multiple (LCM)?
- Is preemption necessary?

Feasible task assignment/allocation

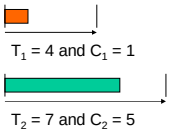


Feasible task assignment/allocation

- It utilizes 2/3 + 1/4 = 11/12 = 0,9167 of the processor capacity if we assume that the LCM is performed as fast as possible (a high sampling rate is often desired)
- It is feasible
- LCM = 12 time units
- Preemption (or cooperative scheduling) is needed
- The spare capacity is 1 time unit or 1/12 of the capacity

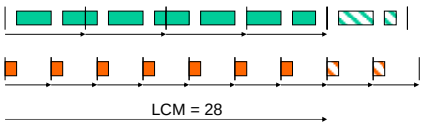
Task Set, Example 1

- This is an example of a task set
- The utilization is 5/7 + 1/4 = 27/28 = 0,9643



Task Set, Example 1

- For this task set, there is one spare unit of capacity in each LCM that not is utilized

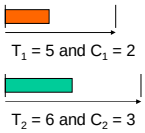


The processor resource

- A processor is a resource that mostly is intended to be used by more than one program
- However, a processor can only execute one program at a time
- A processor is blocked if the program it runs has to wait for an external event (unless it can run some other program while waiting)
- When a program blocks the processor (waiting on an event) we want to leave the program's state where it is and run another program for a while

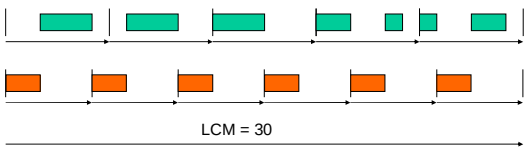
Task Set, Example 2

- This is another example of a task set
- The utilization is 1/2 + 2/5 = 9/10 = 0,90



Task Set, Example 2

- In this case three spare units of capacity are not utilized in each LCM
- This corresponds to the 10% processor capacity that not is used



Worst case execution time

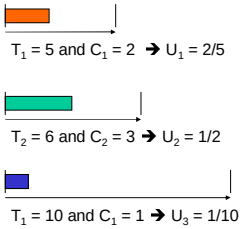
- There is a choice between determinism with regard to data precision and with regard to execution (computation) time
- Algorithms can be written in ways that reduce their variation with regard to their execution time
- Many iterative algorithms terminates when the result does not improve more than a predefined (error) value
- To limit timing variation one can fix the maximum number of iterations, at the cost of reduced or uncertain precision
- We call this Max_time or Max_count limited algorithm iteration loops

WCET uncertainties

- Modern computers have very good throughput and low average computation times for frequently used instructions, branches and contexts
- The variance between different instructions and branches is one the other hand quite large
- Common causes to this uncertainty or time non-determinism are:
  - Pipelining
  - Branch Prediction
  - Cache Hit Ratio
  - Garbage Collection
  - Power Saving Modes
  - Operating System (overhead and service execution times)
  - Compiler (what code is generated?)
- This results in non-deterministic behaviour that make analysis of worst case execution time (WCET) difficult

Task Set, Example 3

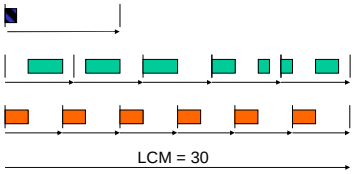
The processor utilization U = 1/2 + 2/5 + 1/10= 1



Is this task set feasible?

Task Set, Example 3

- No, there is no room for the lowest priority task
- However, there is some headroom at the end of the LCM, (allowing a small task with period = LCM)



Find the LCM period

List multiples
of the task periods and look for the smallest common multiple that appears in each list.

Example:

Analyze the task set with T_1 = 4, T_2 = 7

4, 8, 12, 16, 20, 24, **28**, …

7, 14, 21, **28**, … (build lists lazy so that the largest numbers follows each other in size)

Find the LCM period, alternative.

Prime factorization
Prime factorize the task periods of the task set and search for the largest number of times a prime number is used (i.e., the largest power of each prime) and then multiply these prime factors

{4, 6} → $4 = 2^2$ and $6 = 2^1 * 3^1$

→ $2^2 * 3^1 = 12$

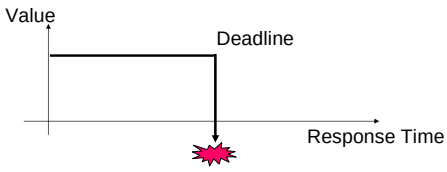
{4, 7} → $4 = 2^2$ and $7 = 7^1$

→ $2^2 * 7^1 = 28$

{5, 6, 10} → $5 = 5^1$, $6 = 2^1 * 3^1$ and $10 = 2^1 * 5^1$

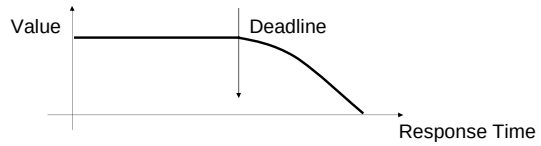
→ $2^1 * 3^1 * 5^1 = 30$

Hard (predictable) real-time



- To avoid catastrophic events a hard real-time system must do its work not only correct but also in time. Data computations must be correct and absolute time limits must be followed, otherwise a system crash with serious consequence could occur.
- A missed deadline is very costly and regarded as a failure in this type of system. Example: An aircraft control system missing a deadline when trying to get out of a spin situation is bad.

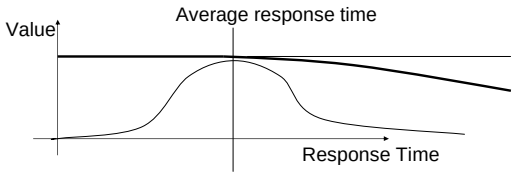
Soft (probabilistic) real-time



In **soft** real-time systems characteristics such as overload robustness, high capacity, high dependability, high throughput or low average response time are more important than guaranteeing deadlines. Time limits are negotiable and can be traded for better average or overall behaviour.

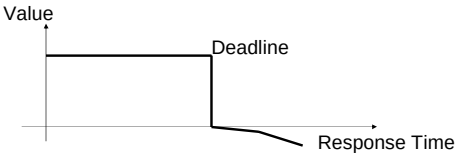
Example: To reject a few new phone calls is better than to risk losing all ongoing calls

Non real-time (best effort)



In best effort systems things such as high throughput and high availability and other aspects on service quality are much more important than timeliness.

Firm real-time



- Value decreases towards zero if deadlines are missed
- Meaning that it might be unavailable until restarted
- Making it less attractive but still acceptable to customers
- No large cost related to accidents or catastrophic events

Real time

- Tasks are done both correct and within time requirements
- Time points when observations and actions are made is handled accurate enough
- To know the time it takes to perform a task is basic to be able to keep its deadline

Real time, cont.

- If we observe a moving object it is equally important to establish its position in space as well as in time
- A common (synchronised) and accurate clock time and known physical space reference points are fundamental
- Time and position stamped samples from two observers can say something about both speed and direction

General Scheduling Principles

- Single processor algorithms give highest priority to tasks with:
  - shortest period
  - closest deadline
  - most part done (in proportion to task size)
  - highest value (reliability or quality related)
  - least recently executed (of equal priority tasks)
- Algorithms for multi, parallel and distributed processor systems also give priority to task with:
  - short slack (deadline - computation time)