Real-Time Systems

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

April 24, 2014

Organization

- Instructor: Jian-Jia Chen, jian-jia.chen@cs.uni-dortmund.de
- Grading: Oral Exam (100%)
- Credit: 6
- Office hour: Wed., 10:30-11:30 AM. Please make appointments.
- `http://ls12-www.cs.tu-dortmund.de/daes/de/lehre/english-courses/ss14-real-time-systems.html`
- References
  - Textbooks:
    - Giorgio C. Buttazzo, “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications”, Springer, Second Edition, 2004. ISBN: 0-387-23137-4
    - Jane W.S. Liu, “Real-Time Systems,” Prentice Hall, First Edition, 2000. ISBN:0130996513
  - Papers from conferences and journals

Course Description

Time: Wednesday, 14:30-16:00, 16:15-1745 weekly
Place: Room 205 at OH16, TU Dortmund
Start: on 23.04.2014
Prerequisite: Computer Operating Systems or equivalent. A basic background in algorithm analysis, data structures, and discrete math will be assumed.
Course material: The slides contain material of the above textbooks, paper references, and course lectures from Steve Goddard, Kevin Jeffay, Nathan Fisher, Lothar Thiele, James Anderson, Kai Lampka, Alan Burns, and Sanjoy Baruah.
Course Calendar:

- The course will last for 12weeks (including this week).

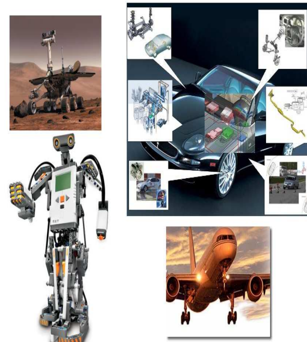
Embedded Systems

Complex "best effort" systems

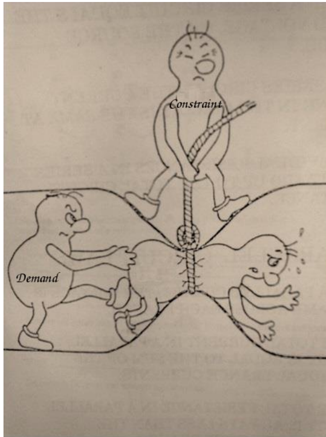
- Mobile telecommunications
- Consumer products, e.g., digital camera, digital video, etc.
- Good reactivity and good dependability

Critical control systems

- Automated aircraft landing systems
- Automotive control for gearing, ABS, airbag, etc.
- High reactivity and high dependability



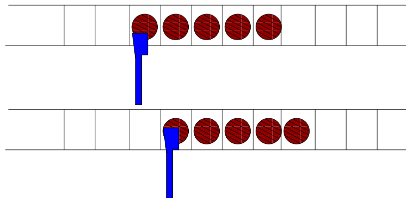
Characteristics of Embedded Systems (Illustrations)



- Even though the hardware improvement continues, the computing demand also increases
- Efficiency
  - Executing should be energy-efficient, code-efficient, and cost-efficient
- Dependability
  - Reliability, maintainability, and availability after deployment
  - Safety even for failure
- Timing guarantee

Real-Time Systems

- Dual notations of correctness:
  - Logical correctness (“the results are correct”)
    - Requires functional analysis
  - Temporal correctness (“the results are delivered in/on time”)
    - Requires non-functional analysis
- High reactivity and high dependability are more important than performance
- **Example:** A robot arm picking up objects from a belt:



Examples for Real-Time Systems

- Chemical & Nuclear Power Plants
- Railway Switching Systems
- Flight Control Systems
- Space Mission Control
- Automotive Systems
- Robotics
- Telecommunications Systems
- Stock Market, Trading System,
- Information Access
- Multimedia Systems
- Virtual Reality
- ......

Hard Real-Time Systems

Catastrophic if some deadlines are missed

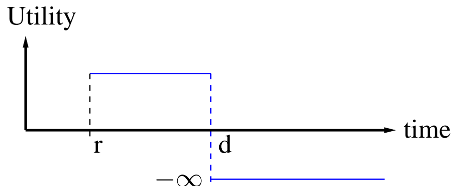
Firmed Real-Time Systems

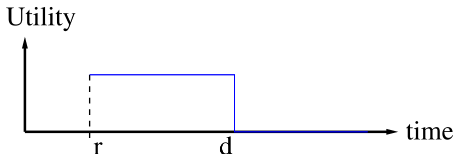
The results are useless if the deadlines are missed

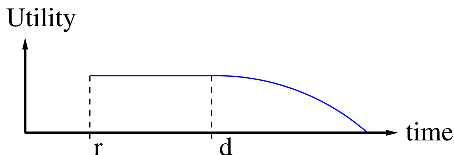
Soft Real-Time Systems

The results are not very useful if the deadlines are missed

Classifications of Real-Time Systems







Hard Real-Time Systems

Catastrophic if some deadlines are missed

Firmed Real-Time Systems

The results are useless if the deadlines are missed

Soft Real-Time Systems

The results are not very useful if the deadlines are missed

Characteristics of Real-Time Systems

- Timeliness
- High cost of failure.
- Concurrency/multiprogramming
- Stand-alone/continuous operation
- Design for worst cases
- Reliability/fault-tolerance requirements
- Predictable behavior

Frequent Misconceptions

- “Real time” is performance engineering/tuning.
  - Timeliness is more important in real-time systems.
- Real-time computing is equivalent to fast computing.
  - Real-time computing means predictable and reliable computing.
- There is no science in real-time system design.
  - Let’s discuss this at the end of the semester.
- Advances in supercomputing hardware will take care of real-time requirements.
  - Buying a ”faster” processor may result in timeliness violation.
- It is not meaningful to talk about guaranteeing real-time performance when things can fail.
  - Though hardwares may fail, the logic components, such as operating systems, should be solid when hardwares are still well functional.

Needs of Concurrency

Reasons for concurrency

- Functional
  - allow multiple users
  - perform many operations concurrently
- Performance
  - take advantage of blocking time
  - parallelism in multi-processor machines
- Expressive Power
  - many control application are inherently concurrent
  - concurrency support helps in expressing concurrency, making application development simpler

Multi-Tasking

- The execution entities (tasks, processes, threads, etc.) are competing from each other for shared resources
- Scheduling decision policy is needed
  - When to schedule an entity?
  - Which entity to schedule?
  - How to schedule entities?

General-Purpose Systems and Real-Time Systems

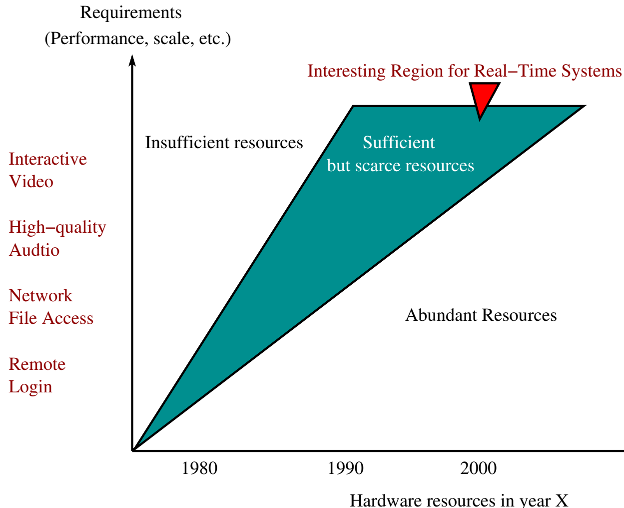
Real-Time Systems

- Applications are known a priori
- Programed by designers
- Timeliness
  - Worst-case response time guarantee
- Examples: traffic control systems, robotics, etc.

General-Purpose Systems

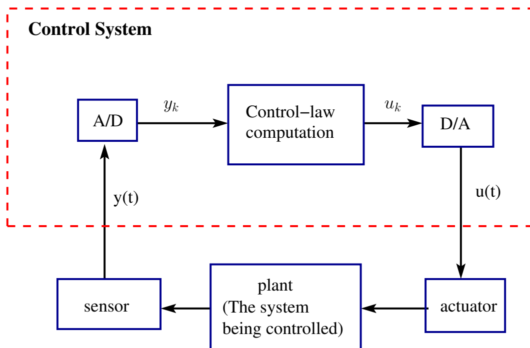
- Broad class of applications
- Programmable by end-users
- The faster, the better
  - Average-case response time
- Examples: Desktops, web servers, etc.

Scarcity of Resources



Example: Simple Control System

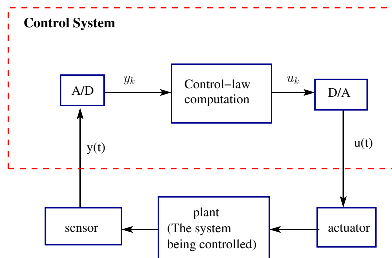
- Many embedded systems are control systems
  - Robotics
  - Automotive Systems
- A simple example: one-sensor and one-actuator control system.



Example: Simple Control System (cont.)

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, which is a key design choice.

Example: Multi-Rate Control System

More complicated control systems have multiple sensors and actuators and must support control loops of different rates.
Example: 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 six cycles 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;

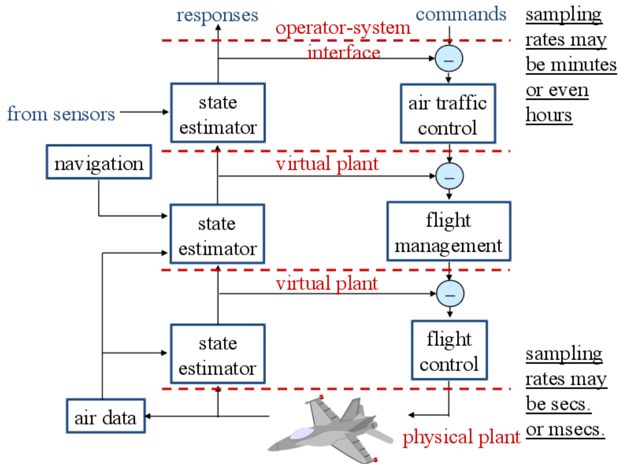
Every other cycle do:

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;

This leads to periodic execution of each control task.

Example: Hierarchical Control System

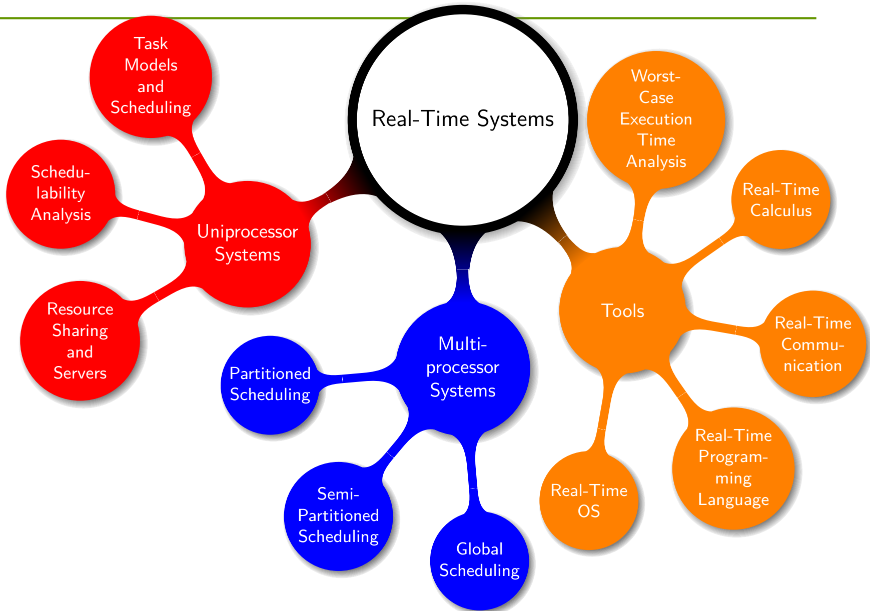


Course Focus

- Basic tools and analytical methods for real-time systems
  - Scheduling algorithms for real-time systems
    - earliest-deadline-first scheduling, rate monotonic scheduling, deadline monotonic scheduling, etc.
    - resource sharing and servers
    - multiprocessor scheduling
  - Basic analysis of timing satisfaction in real-time systems
  - Real-time operating systems, communication and programming languages
- Several advanced topics related to the current research directions, including resource augmentation analysis, mixed criticality, and real-time calculus

After the course, you are expected to know

- fundamental building blocks in real-time systems, and
- schedulability analysis of scheduling algorithms in real-time systems.


Real-Time Systems
Uniprocessor Systems
Task Models and Scheduling
Schedu- lability Analysis
Resource Sharing and Servers
Multi- processor Systems
Partitioned Scheduling
Semi- Partitioned Scheduling
Global Scheduling
Tools
Worst- Case Execution Time Analysis
Real-Time Calculus
Real-Time Commu- nication
Real-Time Program- ming Language
Real-Time OS

Course Calendar

23,04,14	Introduction of Real-Time Systems and Task Models
30,04,14	Analysis of RM and EDF Scheduling (I)
07,05,14	Analysis of RM and EDF Scheduling (II)
14,05,14	Worst-Case Execution Time (WCET)
21,05,14	Resource Sharing and Reservation Servers
28,05,14	Real-Time Programming Languages and Time Automata
04,06,14	Real-Time Operating Systems
11,06,14	Real-Time Communications and Real-Time Calculus
18,06,14	Multiprocessor Scheduling (I)
25,06,14	Multiprocessor Scheduling (II)
02,07,14	Soft Real-Time Systems
09,07,14	Break (conference trip)
16,07,14	Mixed Criticality and Advanced Topics

Real-Time Systems Community

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

- http://tcrts.org/

Fundamentals

- Algorithm:
  - It is the logical procedure to solve a certain problem
  - It is informally specified a sequence of elementary steps that an "execution machine" must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
  - It is the implementation of an algorithm in a programming language
  - It can be executed several times with different inputs
- Process/job/task:
  - An instance of a program that given a sequence of inputs produces a set of outputs

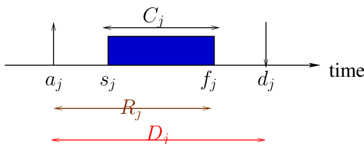
Operating System

An operating system is a program that

- acts as an intermediary between a user of a computer and the computer hardware by providing interfaces
- provides an "abstraction" of the physical machine (for example, a file, a virtual page in memory, etc.)
- manages the access to the physical resources of a computing machine
- makes the computer system convenient to use
- executes user programs and makes solving user problems easier
- and more ......

Timing parameters of a job J_j

- Arrival time (a_j) or release time (r_j) is the time at which the job becomes ready for execution
- Computation (execution) time (C_j) is the time necessary to the processor for executing the job without interruption (= WCET).
- Absolute deadline (d_j) is the time at which the job should be completed.
- Relative deadline (D_j) is the time length between the arrival time and the absolute deadline.
- Start time (s_j) is the time at which the job starts its execution.
- Finishing time (f_j) is the time at which the job finishes its execution.
- Response time (R_j) is the time length at which the job finishes its execution after its arrival, which is $f_j - a_j$.

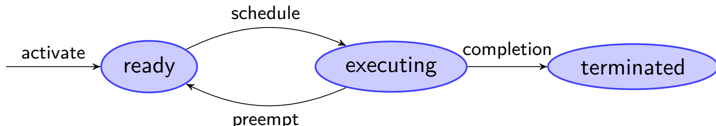


Multi-Tasking (Recap)

- The execution entities (tasks, processes, threads, etc.) are competing from each other for shared resources
- Scheduling policy is needed
  - When to schedule an entity?
  - Which entity to schedule?
  - How to schedule entities?

Scheduling Concepts

- **Scheduling Algorithm**: determines the order that jobs execute on the processor
- Jobs (a simplified version) may be in one of three states:

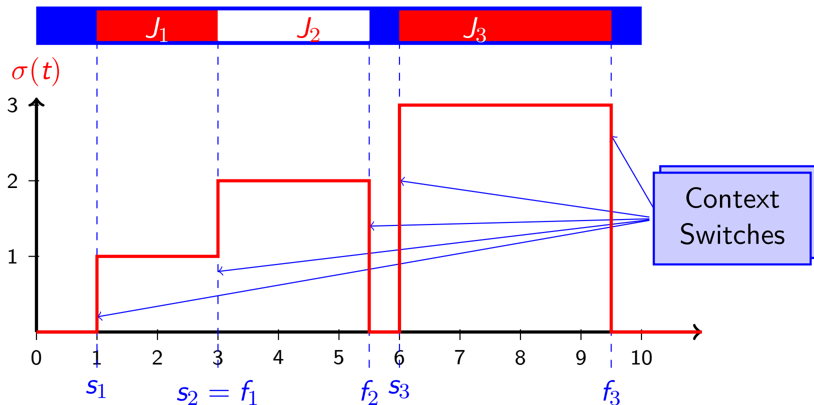


Schedules for a set of jobs $\{J_1, J_2, \ldots, J_N\}$

- A schedule is an assignment of jobs to the processor, such that each job is executed until completion.
- A schedule can be defined as an integer step function $\sigma : \mathbb{R} \rightarrow \mathbb{N}$, where $\sigma(t) = j$ denotes job J_j is executed at time t, and $\sigma(t) = 0$ denotes the system is idle at time t.
- If $\sigma(t)$ changes its value at some time t, then the processor performs a context switch at time t.
- Non-preemptive scheduling: there is only one interval with $\sigma(t) = j$ for every J_j, where t is covered by the interval.
- Preemptive scheduling: there could be more than one interval with $\sigma(t) = j$.

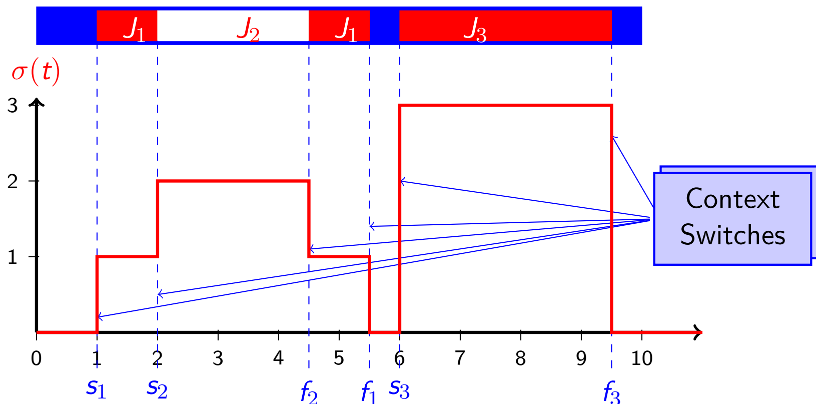
Scheduling Concept: Non-preemptive

Schedule: $\sigma : \mathbb{R} \rightarrow \mathbb{N}$ function of processor time to jobs



Scheduling Concept: Preemptive

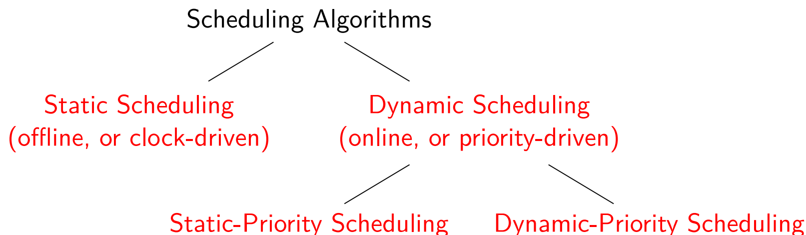
Schedule: $\sigma : \mathbb{R} \rightarrow \mathbb{N}$ function of processor time to jobs



Feasibility of Schedules and Schedulability

- A schedule is feasible if all jobs can be completed according to a set of specified constraints.
- A set of jobs is schedulable if there exists a feasible schedule for the set of jobs.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

Scheduling Algorithms



- Preemptive vs. Non-preemptive
- Guarantee-Based vs. Best-Effort
- Optimal vs. Non-optimal

Evaluating A Schedule

For a job J_j:

- Lateness L_j: delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness E_j: the time that a job stays active after its deadline.

$$E_j = \max\{0, L_j\}$$

- Laxity (or Slack Time)(X_j): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

Metrics of Scheduling Algorithms (for Jobs)

Given a set $\mathbb{J}$ of n jobs, common metrics are to minimize

- Average response time:

$$\sum_{J_j \in \mathbb{J}} \frac{f_j - a_j}{|\mathbb{J}|}$$

- Makespan (total completion time):

$$\max_{J_j \in \mathbb{J}} f_j - \min_{J_j \in \mathbb{J}} a_j$$

- Total weighted response time:

$$\sum w_j(f_j - a_j)$$

- Maximum latency:

$$L_{\max} = \max_{J_j \in \mathbb{J}} (f_j - d_j)$$

- Number of late jobs:

$$N_{late} = \sum_{J_j \in \mathbb{J}} miss(J_j),$$

where $miss(J_j) = 0$ if $f_j \leq d_j$, and $miss(J_j) = 1$ otherwise.

Hard/Soft Real-Time Systems

- Hard Real-Time Systems
  - If any hard deadline is ever missed, then the system is incorrect
  - The tardiness for any job must be 0
  - **Examples**: Nuclear power plant control, flight control
- Soft Real-Time Systems
  - A soft deadline may occasionally be missed
  - Various definitions for "occasionally"
    - minimize the number of tardy jobs, minimize the maximum lateness, etc.
  - **Examples**: Telephone switches, multimedia applications

We mostly consider hard real-time systems in this course.

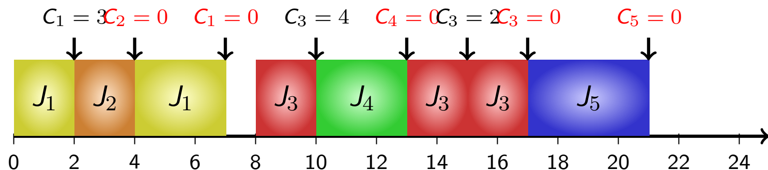
An Example: Shortest-Job-First (SJF)

- At any moment, the system executes the job with the *shortest* remaining time among the jobs in the ready queue.

	J_1	J_2	J_3	J_4	J_5
a_j	0	2	8	10	15
C_j	5	2	6	3	4
d_j	6	8	20	14	22

Exercise

What is the average response time of the above schedule?



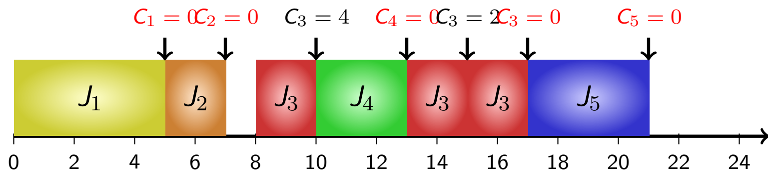
An Example: Earliest-Deadline-First (EDF)

- At any moment, the system executes the job with the *earliest absolute deadline* among the jobs in the ready queue.

	J_1	J_2	J_3	J_4	J_5
a_j	0	2	8	10	15
C_j	5	2	6	3	4
d_j	6	8	20	14	22

Exercise

What is the average response time of the above schedule?

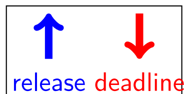


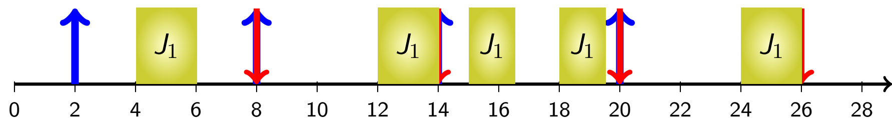
Recurrent Task Models

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time.
- Sporadic Task τ_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i, T_i, D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time.
- Aperiodic Task: Identical jobs released arbitrarily (we will revisit this part in Real-Time Calculus).

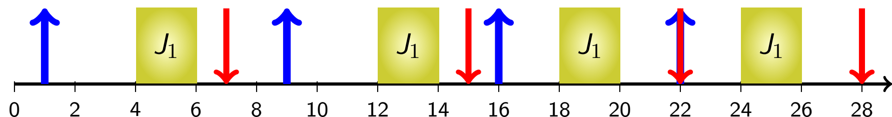
Examples of Recurrent Task Models

Periodic task: $(\phi_i, C_i, T_i, D_i) = (2, 2, 6, 6)$





Sporadic task: $(C_i, T_i, D_i) = (2, 6, 6)$



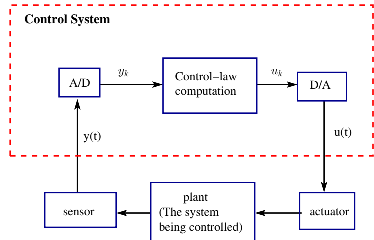
Example: Sporadic Control System

Pseudo-code for this system

while (true)

- start := get the system tick;
- perform analog-to-digital conversion to get y;
- compute control output u;
- output u and do digital-to-analog conversion;
- end := get the system tick;
- $timeToSleep :=$ $T - (end - start)$;
- sleep $timeToSleep$;

end while



Example: Periodic Control System

Pseudo-code for this system

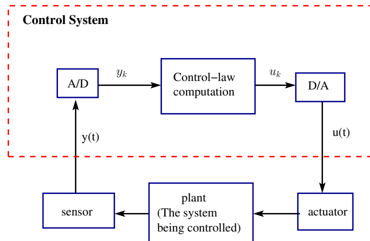
set timer to interrupt periodically with period T;

at each timer interrupt

do

- perform analog-to-digital conversion to get y;
- compute control output u;
- output u and do digital-to-analog conversion;

od



Evaluating A Schedule for Tasks

For a job J_j:

- Lateness L_j: delay of job completion with respect to its deadline.

$$L_j = f_j - d_j$$

- Tardiness E_j: the time that a job stays active after its deadline.

$$E_j = \max\{0, L_j\}$$

- Laxity (or Slack Time)(X_j): The maximum time that a job can be delayed and still meet its deadline.

$$X_j = d_j - a_j - C_j$$

For a task τ_i:

- Lateness L_i: maximum latency of jobs released by task τ_i
- Tardiness E_i: maximum tardiness of jobs released by task τ_i
- Laxity X_i: $D_i - C_i$;

Relative Deadline <=> Period

For a task set, we say that the task set is with

- implicit deadline when the relative deadline D_i is equal to the period T_i, i.e., $D_i = T_i$, for every task τ_i,
- constrained deadline when the relative deadline D_i is no more than the period T_i, i.e., $D_i \leq T_i$, for every task τ_i, or
- arbitrary deadline when the relative deadline D_i could be larger than the period T_i for some task τ_i.

Some Definitions for Periodic Tasks

- The jobs of task τ_i are denoted $J_{i,1}, J_{i,2}, \ldots\ldots$
- Synchronous system: Each task has a phase of 0.
- Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of T_i.
- Task utilization of task τ_i: $u_i = \frac{C_i}{T_i}$.
- System utilization: $\sum_{\tau_i} u_i$.

Feasibility and Schedulability for Recurrent Tasks

- A schedule is feasible if all the jobs of all tasks can be completed according to a set of specified constraints.
- A set of tasks is schedulable if there exists a feasible schedule for the set of tasks.
- A scheduling algorithm is optimal if it always produces a feasible schedule when one exists (under any scheduling algorithm).

Graham's Scheduling Algorithm Classification

- Classification: $a|b|c$
  - a: machine environment
    (e.g., uniprocessor, multiprocessor, distributed, ...)
  - b: task and resource characteristics
    (e.g., preemptive, independent, synchronous, ...)
  - c: performance metric and objectives
    (e.g., $L_{\max}$, sum of finish times, ...)
- Examples:
  - $1|\text{non-prem}|L_{\max}$
  - $M||C_{\max}$

Earliest Due Date Algorithm

Theorem

$1|\text{sync}|L_{max}$: Given a set of n independent jobs that arrive synchronously (release time is 0), any algorithm that executes tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.
Denoted as Earliest Due Date (EDD) Algorithm [Jackson, 1955]

Proof

Let σ be the schedule for J produced by scheduling algorithm A. We can transform A to EDD schedule A' without increasing L_{max}. Details are in the textbook by Buttazzo [Theorem 3.1].

Optimality of EDF

Theorem

Given a set of n independent aperiodic tasks (jobs) with arbitrary arrival times, if the aperiodic task set is feasible on a single processor then any algorithm that executes tasks with earliest deadline (among the set of active tasks) is guaranteed to meet all tasks' deadlines.

- Several proofs of optimality exist: Liu and Layland (1973), Horn (1974), and Dertouzos (1974).
- Similar to Jackson Algorithm proof of optimality, but need to account for preemption.

Monotonicity of Scheduling Algorithms

A good scheduling algorithm should be monotonic

- If a scheduling algorithm derives a feasible schedule, it should also guarantee the feasibility with
  - less execution time of a task/job,
  - less number of tasks/jobs, or
  - more number of processors/machines.

Why is Real-Time Scheduling Hard?

Single-processor (Eisenbrand and Rothvoß, in RTSS 2008)

Fixed-Priority Real-Time Scheduling: Response Time Computation Is $\mathcal{NP}$-Hard

Multiprocessor (Graham 1976)

Changing the priority order, increasing the number of processors, reducing execution times, or weakening precedence constraints can result in a deadline miss.

Many Cases

Scheduling problems in multiprocessor systems are usually $\mathcal{NP}$-Hard.

Fundamentals: Computational Complexity

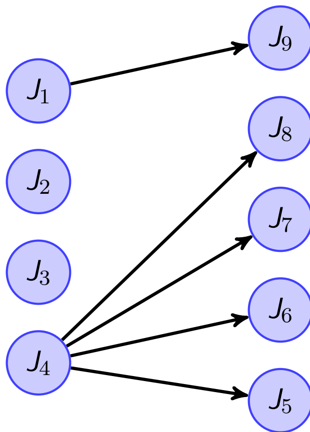
- $\mathcal{N}P$-Complete for a problem Π:
  - If Π can be solved in polynomial time by using a non-deterministic Turning machine, the problem is said in the computational complexity class $\mathcal{N}P$.
  - Π is $\mathcal{N}P$-Complete if Π is in $\mathcal{N}P$ and any problem in the $\mathcal{N}P$ class can reduce to Π in polynomial time (or log space).
  - Π is $\mathcal{N}P$-hard if any problem in the $\mathcal{N}P$ class can reduce to Π in polynomial time (or log space).
- More intuitively (informally)
  - The computing machines we have developed so far are deterministic Turning machines.
  - If Π can be solved in polynomial time by using a deterministic Turning machine, the problem is said in the computational complexity class $\mathcal{P}$.
  - If a problem is $\mathcal{N}P$-Complete or $\mathcal{N}P$-hard, there is no efficient (polynomial-time) algorithm to derive optimal/feasible solutions unless $\mathcal{P} = \mathcal{N}P$.

Multiprocessor Anomalies

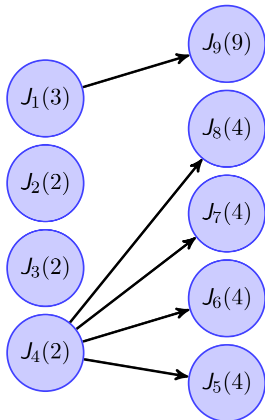
- Partitioned scheduling (Each task/job is on a processor)
  - As most partitioning algorithms are not optimal, a system might become infeasible with
    - Less execution time of a task/job
    - Less number of tasks/jobs
    - More number of processors/machines
- Global scheduling
  - As most priority-assignment algorithms are not optimal, a system might become infeasible with
    - Less execution time of a task/job
    - Less number of tasks/jobs
    - More number of processors/machines

Precedence Constraints

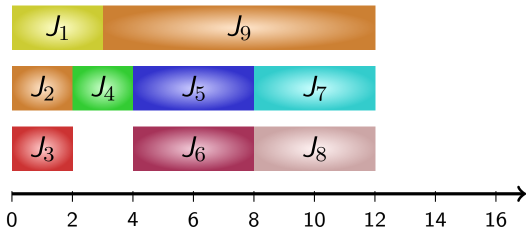
Jobs (and tasks) may have to execute in a pre-specified order.



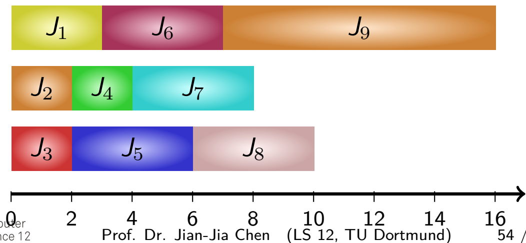
Multiprocessor Anomaly: Case 1



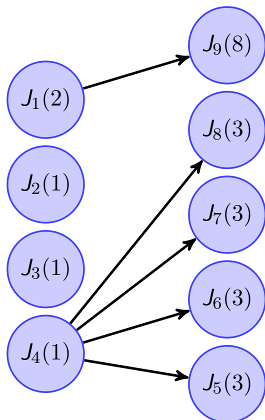
On 3 processors

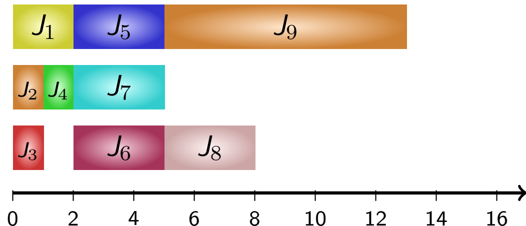


Removing the precedence constraints on J_4...



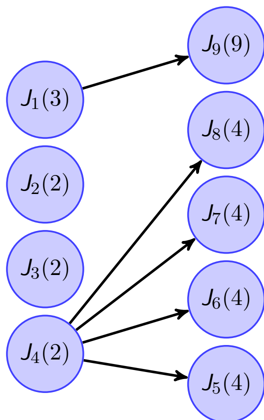
Multiprocessor Anomaly: Case 2



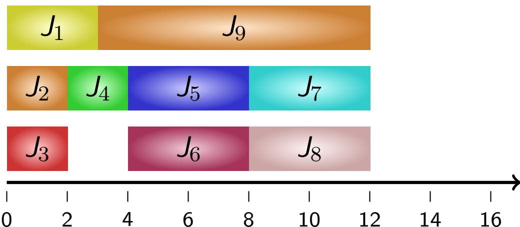


Reduce the execution time by 1, and schedule on 3 processors

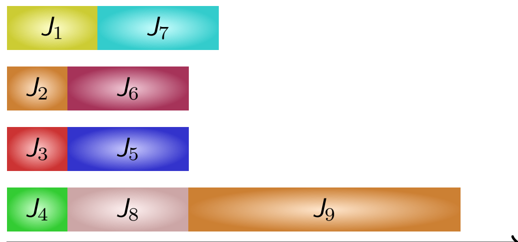
Multiprocessor Anomaly: Case 3



On 4 processors



Use 4 processors



Note about the Material in the Course

- For the rest of the course, we will not cover systems with the following characteristics
  - non-preemptive scheduling or
  - tasks (jobs) with precedence constraints (namely, one job has to wait until another job finishes).
- This means that we will cover the design and analysis for scheduling algorithms with
  - preemptive scheduling and
  - independent tasks/jobs

The material this week has covered the corresponding contents in Chapters 2 and 3 in Buttazzo's textbook.