Clock-Driven Scheduling

IL2212 Embedded Software

Ingo Sander and Matthias Becker

Royal Institute of Technology
Stockholm, Sweden
ingo@kth.se, mabecker@kth.se

Outline

Clock-Driven Scheduling

Clock-driven (or time-driven or timeline) scheduling

- requires a large amount of determinism
- enables to implement efficient schedule with a low overhead, since the schedule can be calculated off-line.
  - complex algorithms can be used
  - amount of processor time allocated to each job is equal to its Worst-Case Execution Time (WCET)
  - static schedule guarantees that every job completes by its deadline as long as no job overruns

Clock-Driven Scheduling

Assumptions

1. There is a constant number of n periodic tasks in the system
2. The parameters of all periodic tasks are known a priori
   - Variations in inter-release times of jobs are negligibly small
   - Each job of τ_i is released T_i units of time after the previous job of τ_i
   - Each job $\tau_{i,k}$ is ready for execution at its arrival time $a_{i,k}$

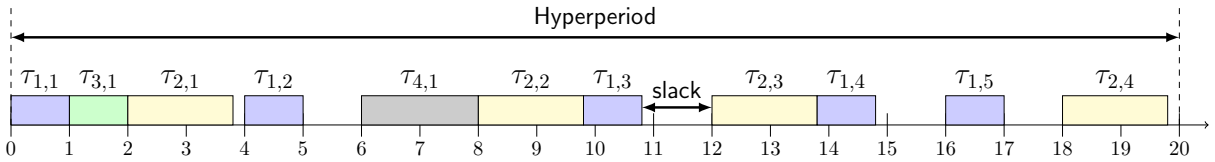
Clock-Driven Scheduling

Example

Four independent periodic tasks: $\tau_1 = (4, 1), \tau_2 = (5, 1.8), \tau_3 = (20, 1), \tau_4 = (20, 2)$

- Utilisation $U = \frac{1}{4} + \frac{1.8}{5} + \frac{1}{20} + \frac{2}{20} = 0.76$
- Hyperperiod $H = LCM(4, 5, 20, 20) = 20$

Possible schedule:

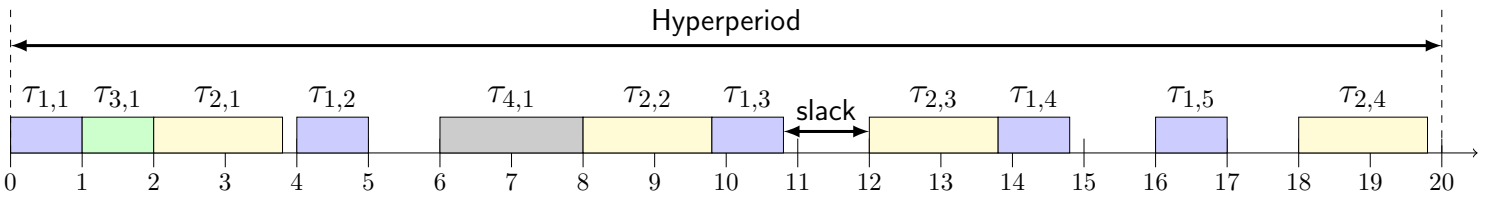


Slack can be used for aperiodic jobs!

Implementing a Cyclic Scheduler

- Store precomputed schedule as table
- Each entry $(t, \tau_k(t))$ in the table gives a decision time t_k at which a scheduling decision is made
- $\tau_k(t)$ can be either a task τ_i or I (idle)
- Idle time can be used for aperiodic jobs

k	t	$\tau_k(t)$
0	0	τ_1
1	1	τ_3
2	2	τ_2
3	3.8	I
4	4	τ_1
5	5	I
6	6	τ_4
7	8	τ_2
8	9.8	τ_1
9	10.8	I
10	12	τ_2
11	13.8	τ_1
12	14.8	I
13	16	τ_1
14	17	I
15	18	τ_2
16	19.8	I



Outline

Aperiodic Jobs

- Aperiodic jobs are released at unexpected time instants.
- Assumptions for following discussion:
  - Aperiodic job have no or soft deadline
  - Aperiodic jobs are placed in special queue
  - New jobs are added to the queue without need to notify scheduler
  - When processor is available aperiodic jobs are scheduled

Aperiodic Jobs

- Aperiodic jobs are released at unexpected time instants.
- Assumptions for following discussion:
  - Aperiodic job have no or soft deadline
  - Aperiodic jobs are placed in special queue
  - New jobs are added to the queue without need to notify scheduler
  - When processor is available aperiodic jobs are scheduled

Aperiodic Jobs and Sporadic Jobs

- **?** states that aperiodic jobs can have no, soft, firm or hard deadline. Aperiodic jobs with a hard deadline are called sporadic jobs, which requires that there is a known minimum interarrival time between sporadic jobs.
- **?** state that aperiodic jobs have no hard deadline. Jobs with a hard deadline are called sporadic jobs (no distinction between jobs with firm and hard deadline). No requirement on interarrival time.

Implementing a Cyclic Scheduler

- Initialisation
  - Tasks to be executed are created and sufficient memory is allocated
  - Code executed by the tasks is loaded into memory
- Scheduler is invoked on hardware timer interrupt
  - First interrupt at $t_k = 0$
  - On receipt of an interrupt
    - Set next timer interrupt to t_{k+1}
    - If $\tau(t_k) = I$ and aperiodic job waiting, then start aperiodic job
    - otherwise schedule next job in task (and preempt aperiodic job, if there is one!)

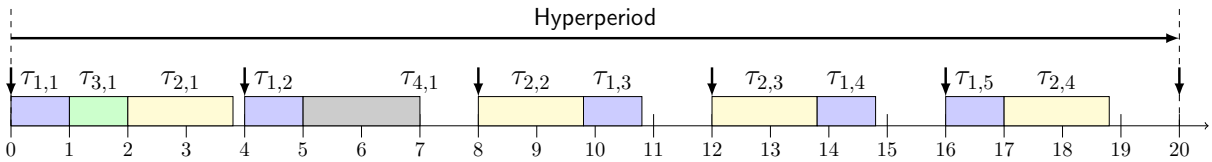
Structure of Cyclic Schedules

Ad hoc table-driven schedules are flexible, but not efficient

- relies on accurate timer interrupts and exact execution times of tasks
- large scheduling overhead
- intervals for aperiodic jobs are not spread out uniformly and may be very short
- interval timer needed

Frame-Based Approach

- make scheduling decision periodically at certain intervals (frames)
- execute a fixed number of jobs in every frame
- each frame has a size of f time units
- preemption is only allowed at frame borders
- the first job of every task is released at the beginning of a frame



The frame is also called Minor Cycle and the hyperperiod is called Major Cycle.

Benefits of Frame-Based Cyclic Schedules

- At the beginning of each frame
  - scheduler can check, if every job in frame has been released and is ready for execution
  - scheduler can detect if there is any overrun or a missed deadline
- Periodic timer instead of hardware timer can be used

Cyclic Executive

The term cyclic executive for a table-driven cyclic scheduler for all types of jobs in a multithreaded system

- Scheduling decisions are made at the beginning of each frame, triggered by timer interrupts
- During execution table entry for current frame is copied into current block
- Scheduler wakes up a task 'periodic task server' that executes all job slices in the current block
- Then scheduler uses remaining time in frame for aperiodic jobs

Efficient Scheduling of Aperiodic Jobs

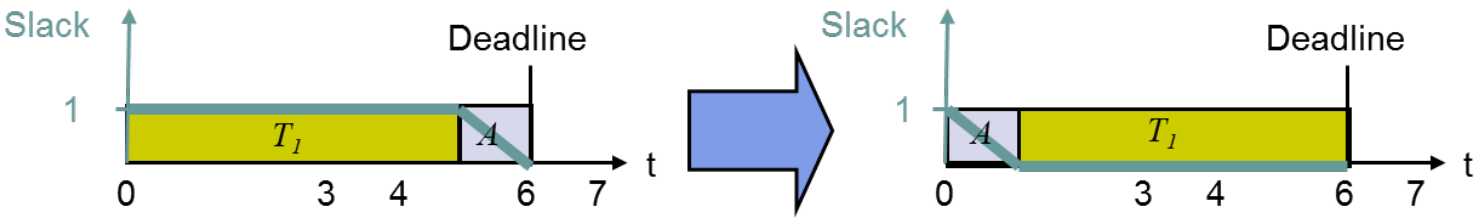
- So far aperiodic jobs have been scheduled in the background after all other job slices have been completed.

⇒ Disadvantage: Average response time is long

- Average response time for aperiodic jobs can be improved by scheduling hard-real time jobs as late as possible without missing the deadline

Slack Stealing

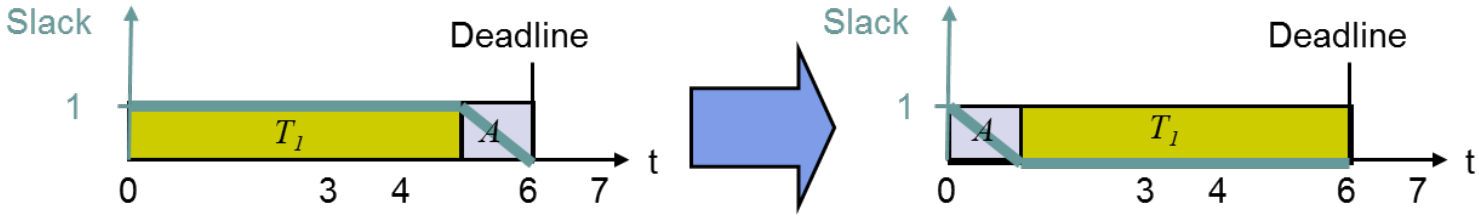
- **Idea:** Use slack to schedule aperiodic jobs before periodic jobs whenever possible!
- **Implementation:** Cyclic executive keeps track of slack and lets periodic task server execute aperiodic jobs as long as there is slack available



Slack Stealing

Interval timer is used

- At beginning of frame timer is set to slack in frame
- Whenever an aperiodic job executes slack is reduced
- When timer expires, slack is consumed and aperiodic job is preempted



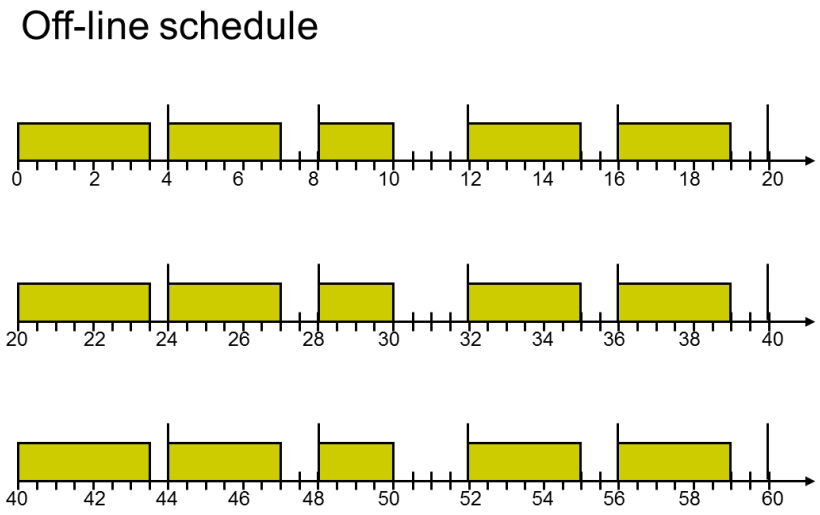
Outline

Scheduling Aperiodic Jobs with Firm Deadlines

- Sporadic tasks have hard deadlines, which requires a known minimum interarrival time between two sporadic jobs. Not discussed here!
- Discussion focuses on scheduling of aperiodic jobs with firm deadline.
  - Release times and maximum execution times are unknown a priori
- Properties of aperiodic job with firm deadline is known at release time.
  - Acceptance test (at start of frame):
    - Aperiodic job with firm deadline is only scheduled, if all scheduled jobs still meet their deadlines
    - Otherwise it is rejected
  - Accepted aperiodic jobs with firm deadline can be scheduled using EDF

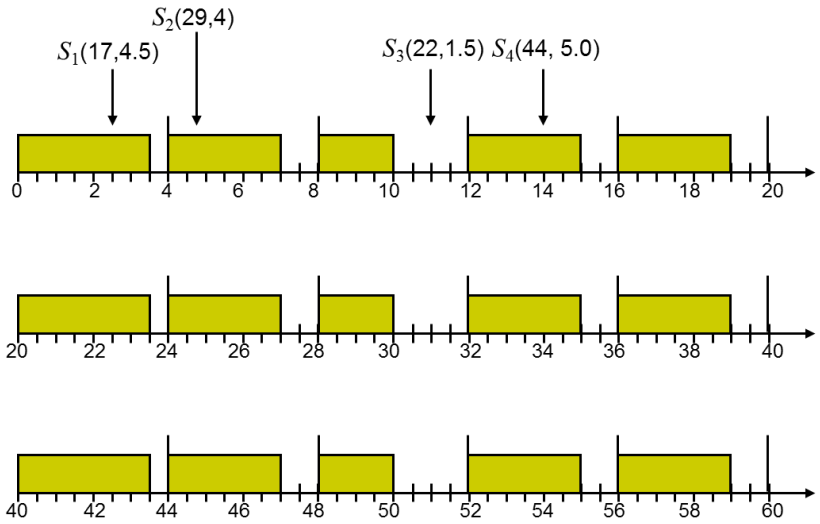
Scheduling Aperiodic Jobs with Firm Deadline

Example



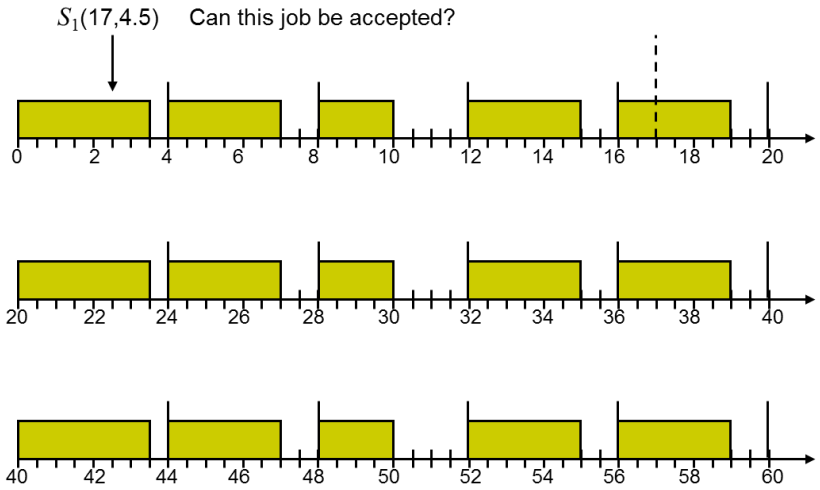
Scheduling Aperiodic Jobs with Firm Deadline

Example



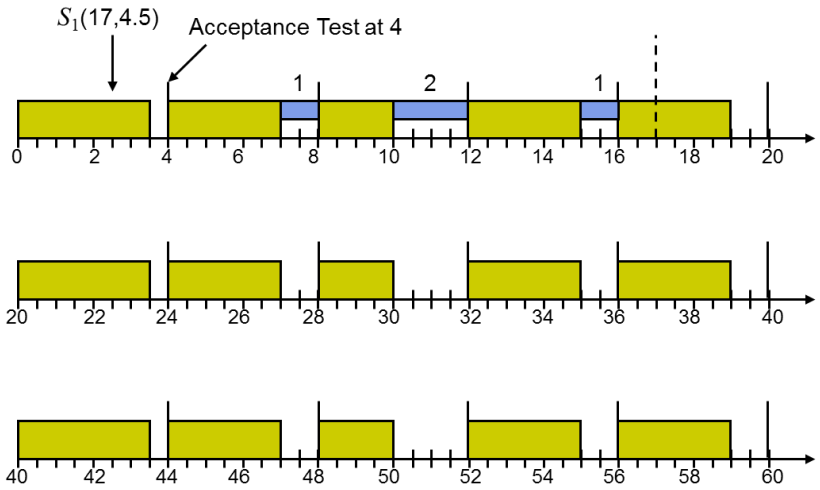
Scheduling Aperiodic Jobs with Firm Deadline

Example



Scheduling Aperiodic Jobs with Firm Deadline

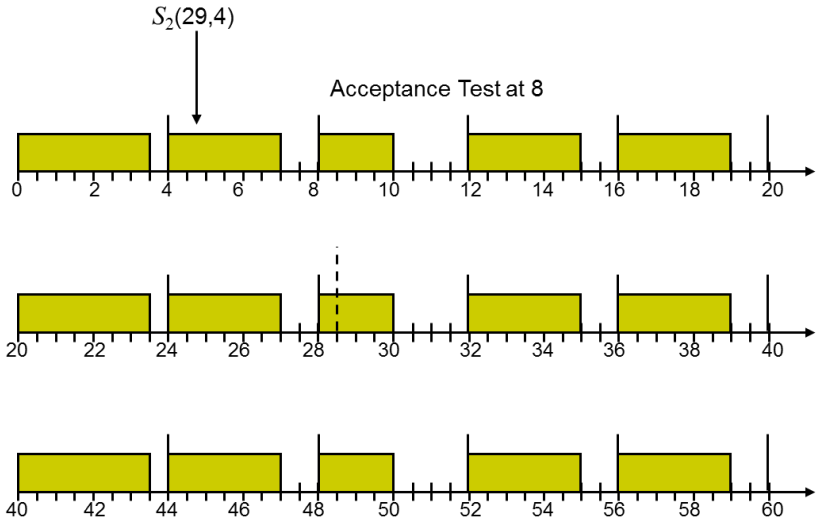
Example



Job rejected: Not sufficient slack (Slack = 4)!

Scheduling Aperiodic Jobs with Firm Deadline

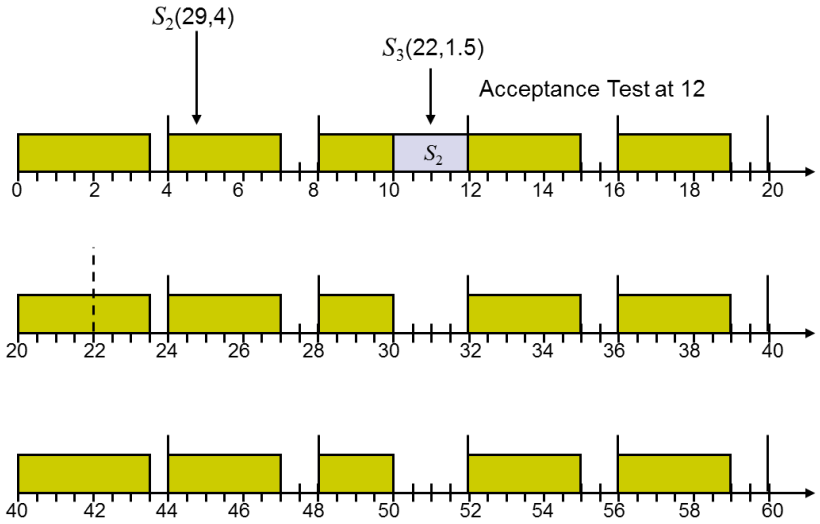
Example



Job S_2 accepted, put into job queue for jobs with firm deadline

Scheduling Aperiodic Jobs with Firm Deadline

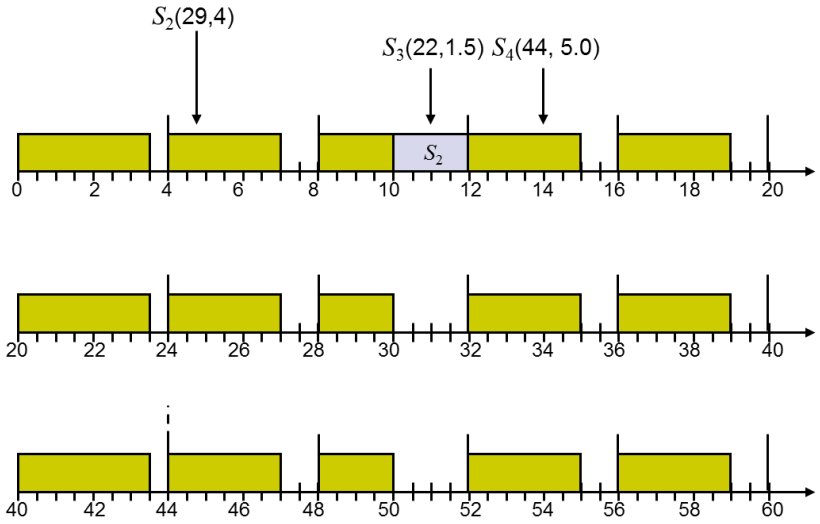
Example



Job S_3 accepted, put before S_2 into for jobs with firm deadline

Scheduling Aperiodic Jobs with Firm Deadline

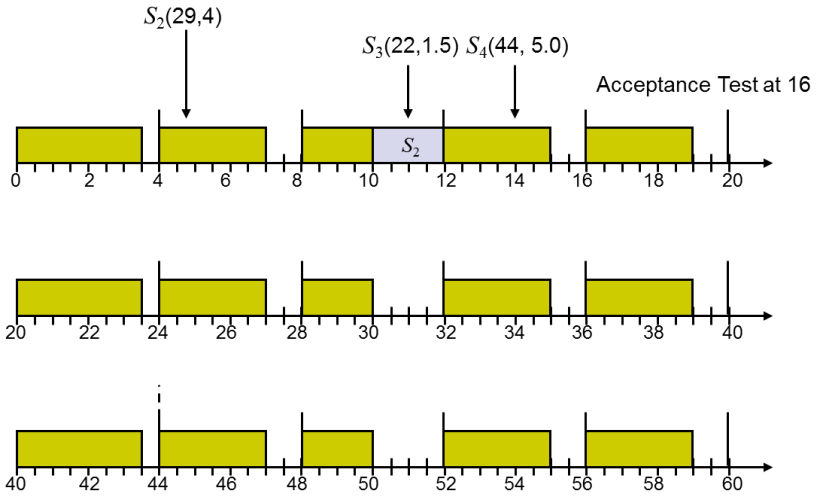
Example



Can job S_4 be accepted?

Scheduling Aperiodic Jobs with Firm Deadline

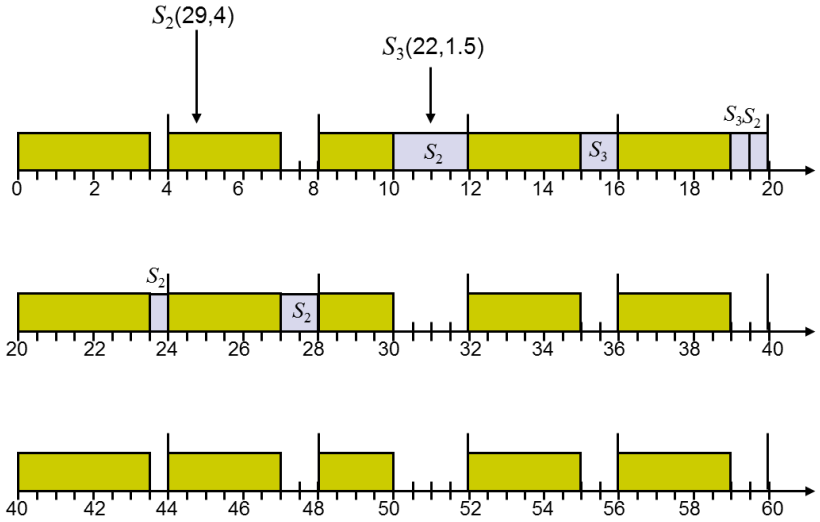
Example



Job rejected: Not sufficient slack!

Scheduling Aperiodic Jobs with Firm Deadline

Example



Practical Considerations

Handling Frame Overruns (Soft Real-Time)

- Overruns may occur, when jobs execution time is longer than maximum execution time
- Can be handled by
  - abort the overrun job and report the premature termination of the job
  - unfinished portion may execute as aperiodic job in a later frame
  - continue to execute the overrun job but this may cause other jobs to be late, too!

Outline

Multiprocessor Systems

Clock-driven scheduling works also for multi-processor systems if all details about application and platform are known and can be specified

- Constructing a feasible schedule for multiprocessors is more complex and time consuming than for uniprocessors
  - The possibility to select the core to execute on adds an additional dimension to the problem
- Since it is done off-line, exhaustive and complex heuristic algorithms can be used

Conclusion Clock-Driven Scheduling

Pros and Cons

Pros

- Conceptual simplicity
  - complex dependencies, communication delays can be considered when developing the schedule
  - no need for any synchronisation mechanisms
  - schedule can be represented as table that is used by the scheduler at run time
- Relatively easy to validate

Conclusion Clock-Driven Scheduling

Pros and Cons

Cons

- Inflexible, since schedule is computed off-line, small changes mean that new tables have to be generated
- Release times of jobs must be fixed
- A lot information about jobs has to known beforehand, so that the schedule can be pre-computed
- Large task sets can result in large table which needs to be stored in memory

Outline

References

Giorgio C. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer, 3rd edition, 2011.

Jane W. S. Liu. Real-Time Systems. Prentice Hall, 2000.