Clock-Driven Scheduling

Ingo Sander

ingo@kth.se

Liu: Chapter 5

Outline

- Clock-driven scheduling
  - Notations and assumptions
  - Static, timer-driven cyclic schedules
  - Handling aperiodic jobs and sporadic jobs
  - Practical considerations
  - Pros and Cons

Determinism

- Clock-driven scheduling requires a large amount of *determinism*
  - only few aperiodic and sporadic jobs
- Schedule can be calculated off-line

Assumptions

1. There is a constant number 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 in T_i is released p_i units of time after the previous job in T_i
3. Each job $J_{i,k}$ is ready for execution at its release time $r_{i,k}$

Assumptions

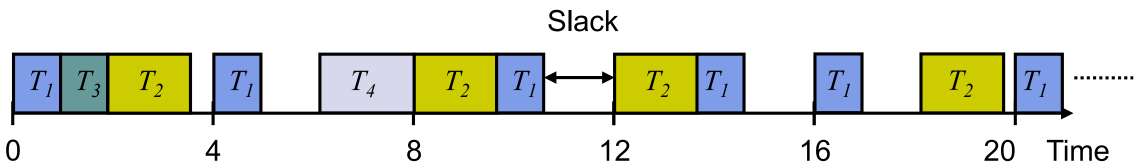
- There are aperiodic jobs released at unexpected time instants
  - aperiodic jobs are placed in special queue
  - new jobs are added to the queue without need to notify scheduler
  - when processor is available aperiodic jobs are scheduled
- There are no sporadic jobs (this assumption will be relaxed later)

Static, Clock-Driven Scheduler

- Static schedule can be calculated off-line (all parameters are known at start)
  - complex algorithms can be used
  - amount of processor time allocated to each job is equal to its maximum execution time
  - static schedule guarantees that every job completes by its deadline as long as no job overruns

Example

- Four independent periodic tasks: $T_1 = (4,1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$
- Utilization = 1/4 + 1.8/5 + 1/20 + 2/20 = 0.76
- Hyperperiod = LCM (4, 5, 20, 20) = 20
- Possible schedule:



Slack time can be used to execute aperiodic jobs

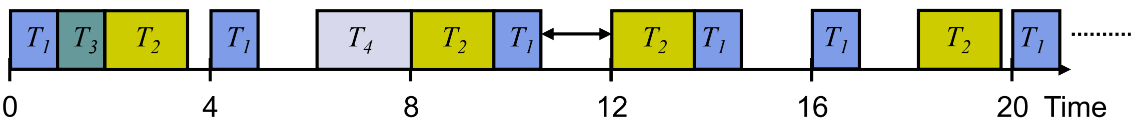
Implementing a Cyclic Scheduler

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

Implementing a Cyclic Scheduler

k	t	$T_k(t)$)
0	0	T_1
1	1	T_3
2	2	T_2
3	3.8	I
4	4	T_1
5	5	I
6	6	T_4
7	8	T_2
8	9.8	T_1

k	t	$T_k(t)$)
9	10.8	I
10	12	T_2
11	13.8	T_1
12	14.8	I
13	16	T_1
14	17	I
15	18	T_2
16	19.8	I



Implementing a Cyclic Scheduler

- Initialization
  - 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 $T(t_k) = I$ and aperiodic job waiting 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 maybe very short
  - interval timer needed

Structure of Cyclic Schedules

- 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
  - preemption is only allowed at frame borders
  - the first job of every task is released at the beginning of a frame

Structure of cyclic schedules

- Benefits
  - 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

For the theory how to calculate the optimal frame size check Liu, Section 5.3 (not part of the course)

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

Cyclic Executive

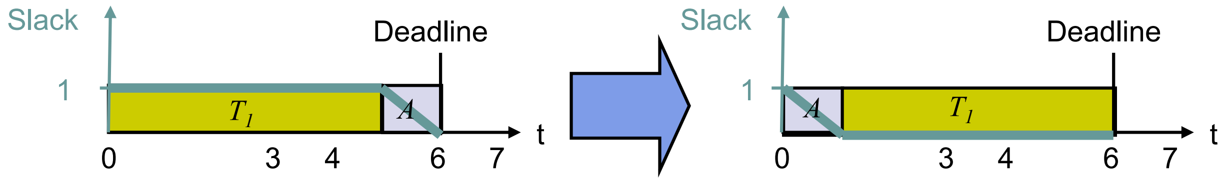
- During execution table entry for current frame is copied into current block
- Scheduler wakes up a job “periodic task server” that executes all job slices in the current block
- Then scheduler uses remaining time in frame for aperiodic jobs

Scheduling 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

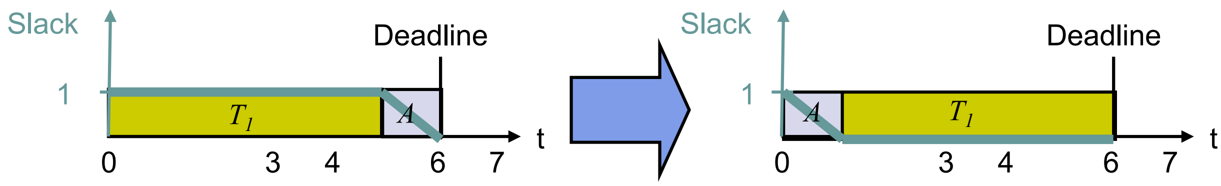
Slack Stealing

- 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



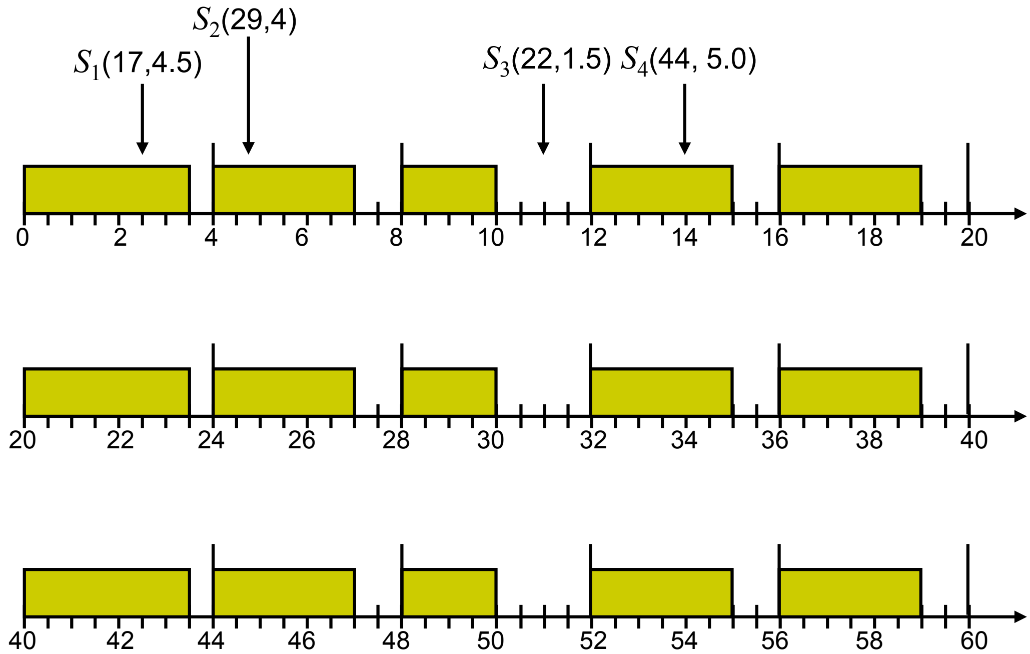
Scheduling Sporadic Jobs

- Sporadic jobs have hard deadlines
- Minimum release times and maximum execution times are unknown a priori
  - No possibility to guarantee deadlines, when calculating schedule off-line before system start

Scheduling Sporadic Jobs

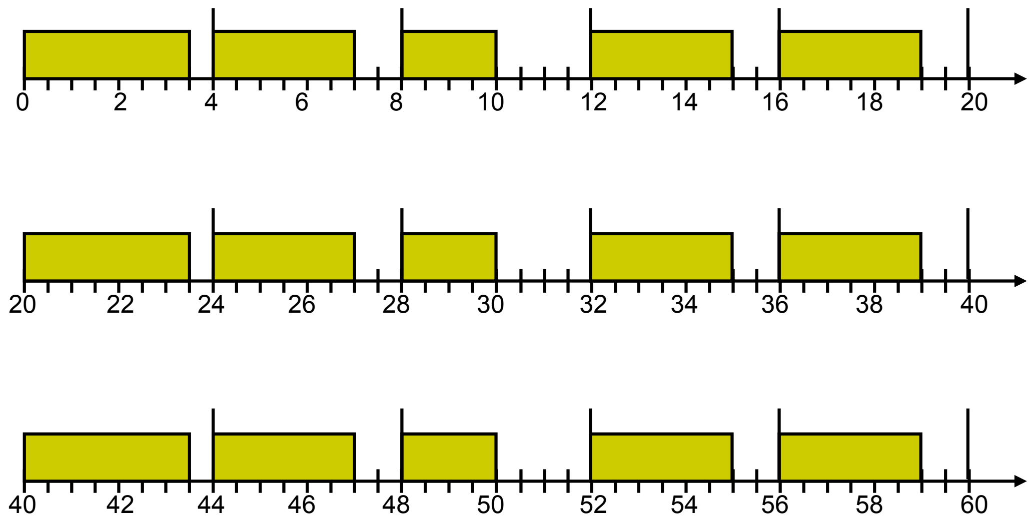
- However the properties of a sporadic job are known, when the sporadic job is released
  - Acceptance test (at start of frame):
    - Sporadic job is only scheduled, if all scheduled jobs still meet their deadlines
    - Otherwise it is rejected
  - Accepted sporadic jobs can be scheduled using EDF

Example
Scheduling Sporadic Jobs

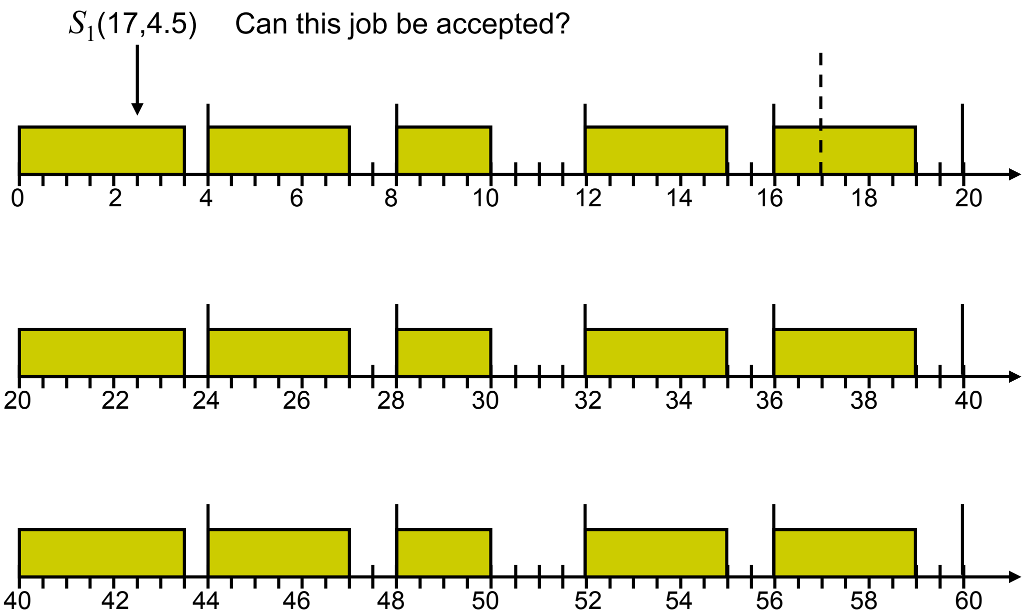


Example
Scheduling Sporadic Jobs

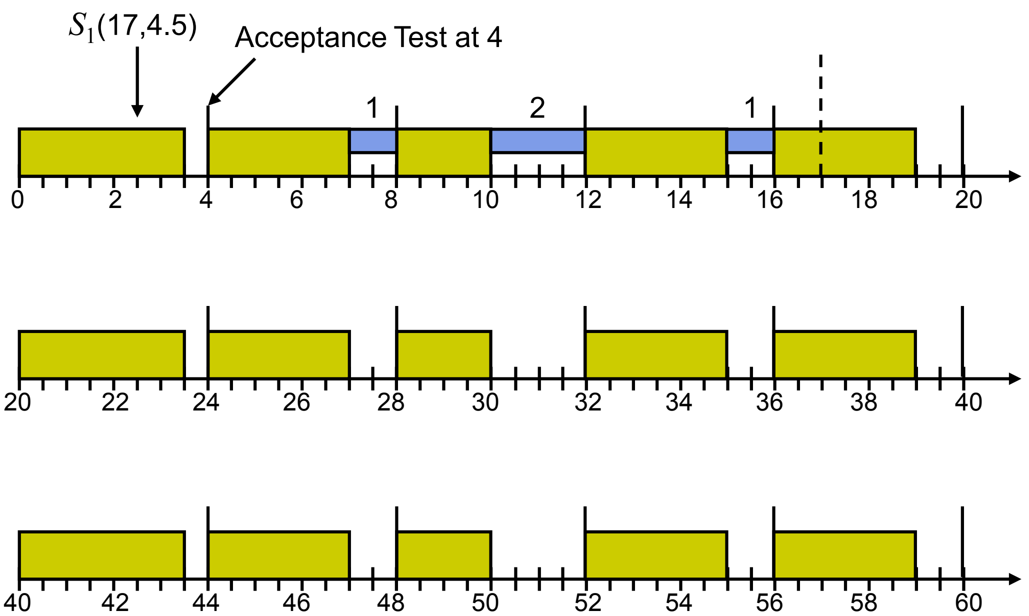
Off-line schedule



Example
Scheduling Sporadic Jobs

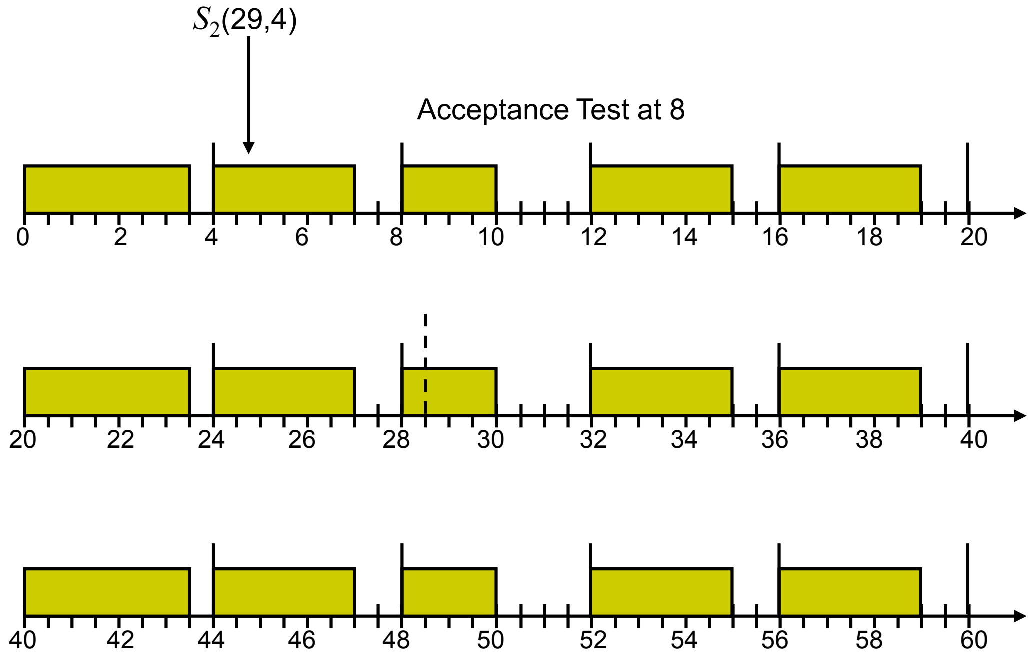


Example
Scheduling Sporadic Jobs



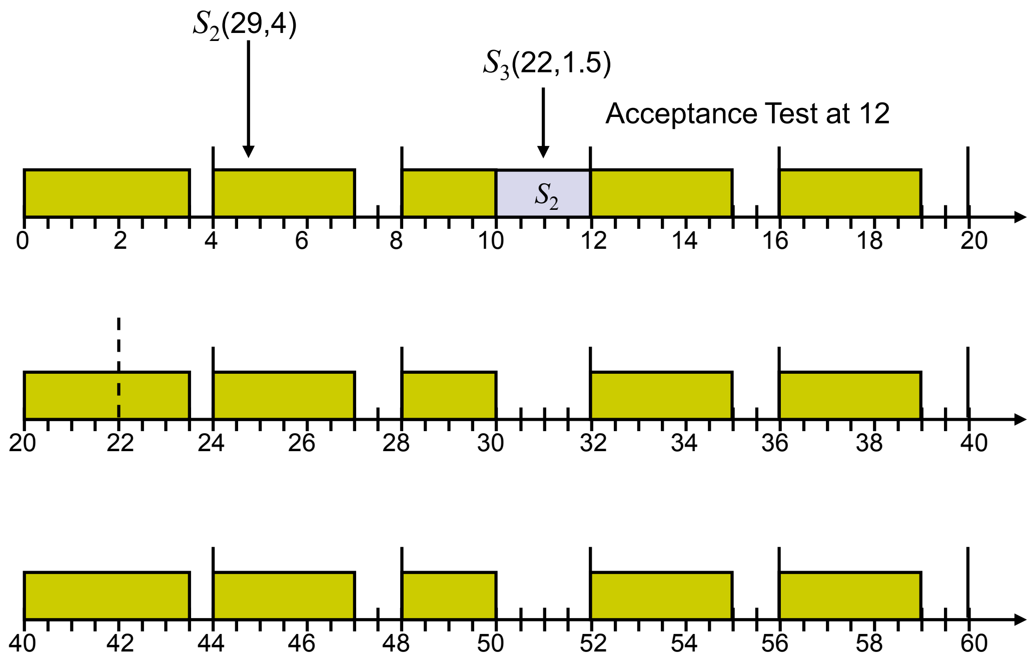
Job rejected: Not sufficient slack (Slack = 4)!

Example
Scheduling Sporadic Jobs



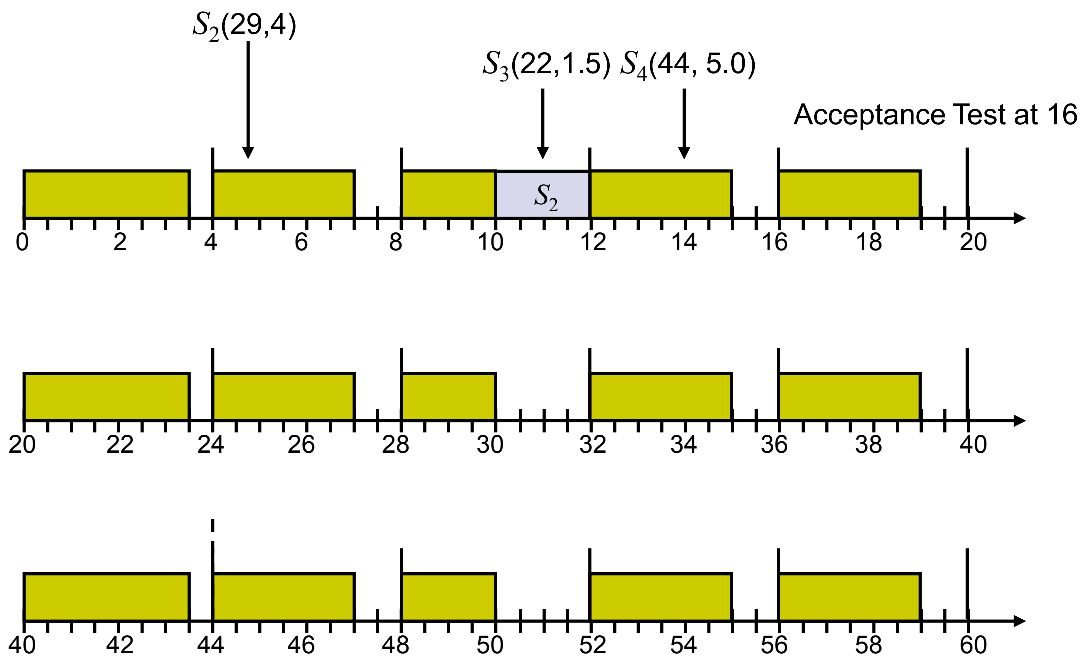
Job S_2 accepted, put into sporadic job queue

Example
Scheduling Sporadic Jobs



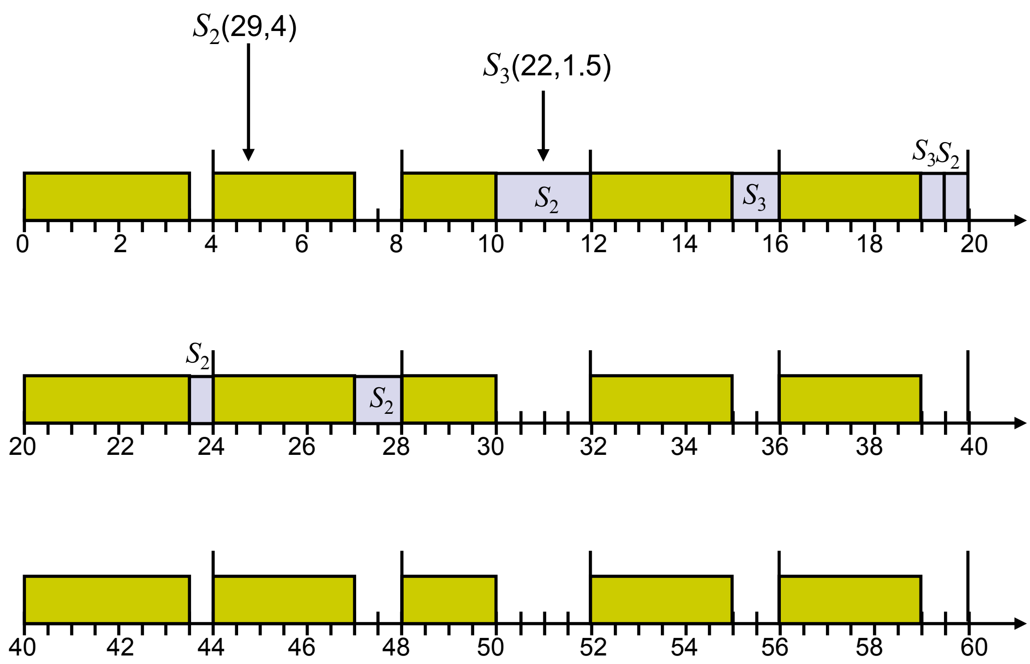
Job S_3 accepted, put before S_2 into sporadic job queue

Example
Scheduling Sporadic Jobs



Job rejected: not enough slack!

Example
Scheduling Sporadic Jobs



Practical Considerations

- Handling Frame Overruns
  - 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!

Practical Considerations

- Multiprocessor system
  - Constructing a feasible schedule for multiprocessors is more complex and time consuming than for uniprocessors
  - Since it is done off-line, exhaustive and complex heuristic algorithms can be used

Advantages

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

Disadvantages

- 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
- Difficult to get acceptable response times for aperiodic (soft real-time) jobs

Summary

- Clock-driven schedulers schedule periodic tasks according to a cyclic schedule
- Task parameters must be known in advance
- Schedule can be calculated in advance
- Aperiodic and sporadic jobs can be scheduled, if they do not influence other scheduled jobs
- Applicable to static systems, with a small number of aperiodic jobs