KTH ROYAL INSTITUTE
OF TECHNOLOGY

Scheduling Aperiodic and Sporadic Tasks in Priority-Driven Systems

Ingo Sander and Matthias Becker
Contact: ingo@kth.se

Liu: Chapter 7
Buttazzo: Chapter 5

IL2212 – Embedded Software

1

Outline

- System Model and Assumptions
- Scheduling Aperiodic Tasks (with soft deadlines)
- Scheduling Aperiodic Tasks (with firm deadlines)

2

Outline

- System Model and Assumptions
- Scheduling Aperiodic Tasks (with soft deadlines)
- Scheduling Aperiodic Tasks (with firm deadlines)

Assumptions

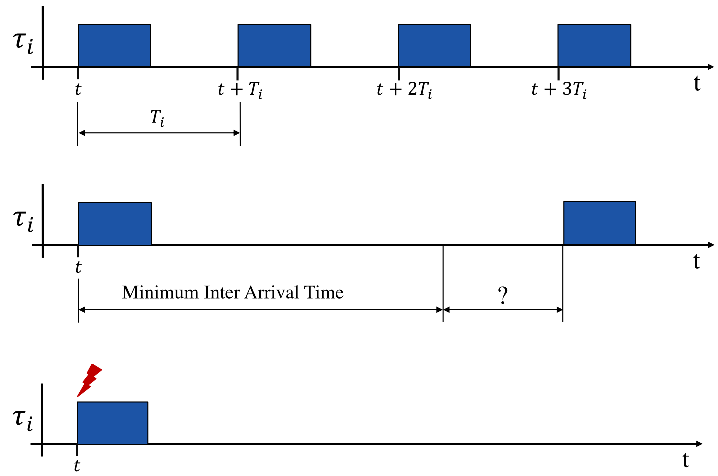
- Single Processor
- Independent preemptable periodic tasks
- Parameters of all periodic tasks are known
- Periodic tasks meet their deadlines
- Aperiodic and sporadic tasks are independent of each other
- Parameters of aperiodic requests with firm deadlines become known after release

System Model

Periodic:

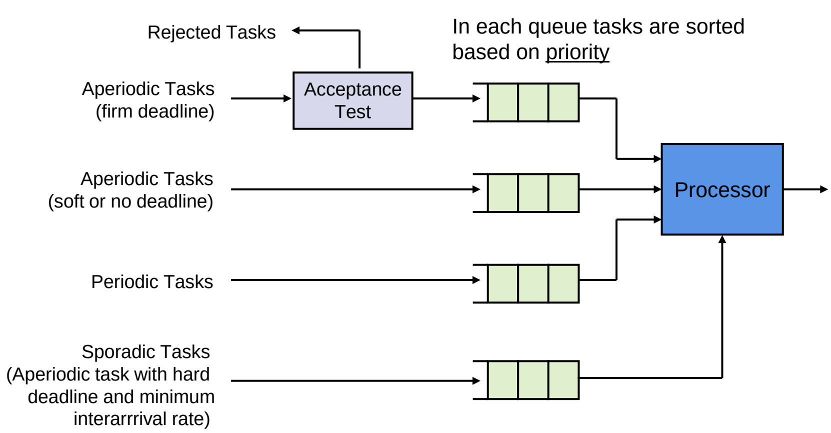
Sporadic:

Aperiodic:



5

System Model



6

Scheduling Algorithms

- Aperiodic job requests with soft deadlines
  - are always accepted
  - Scheduler tries to complete aperiodic jobs as soon as possible

Scheduling Algorithms

- Aperiodic job requests with firm deadlines
  - Scheduler decides, if job can be accepted or must be rejected
    - Job is accepted and scheduled, if all other scheduled jobs still meet their deadlines
    - Otherwise, job is rejected
- Sporadic tasks
  - A sporadic task is an aperiodic task with a *hard* deadline
  - Additional information in form of a *minimum interarrival time* as discussed by Buttazzo [But11] is needed!
- Discussion
  - A sporadic job in [Liu00] is an aperiodic job with hard of firm deadline. Lecture uses Buttazzos definition.

Scheduling Algorithm

- A scheduling algorithm is *correct*, if it only produces correct schedules of the system
- A *correct schedule* is a schedule where periodic tasks and accepted aperiodic jobs with firm deadlines always meet their deadlines

Optimality of Algorithms

- An aperiodic task scheduling algorithm is optimal if it minimizes either
  - the *response time* of the aperiodic task at the head of the aperiodic task queue, or
  - the *average response time* of all the aperiodic tasks for the given queueing discipline

Optimality of Algorithms

- A scheduling algorithm for aperiodic tasks with firm deadline is optimal, if it
  – accepts each aperiodic job with firm deadline newly offered to the system and schedules the task to complete in time *if and only if* the new task can be correctly scheduled in time by some means

Outline

- System Model and Assumptions
- Scheduling Aperiodic Tasks (with soft deadlines)
  - Main principles
  - Background Execution
  - Periodic Server
  - Bandwidth Preserving Server
- Scheduling Aperiodic Tasks (with firm deadlines)

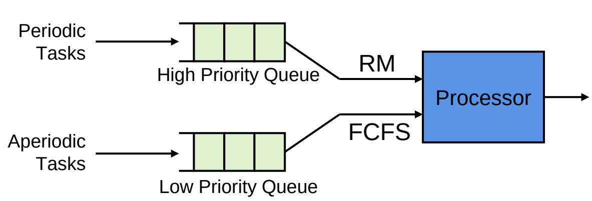
Scheduling Aperiodic Jobs with Soft Deadline

- In the following several different algorithms to schedule aperiodic jobs are discussed
- Assumption here:
  – No sporadic tasks or aperiodic tasks with firm deadline

13

Background Execution

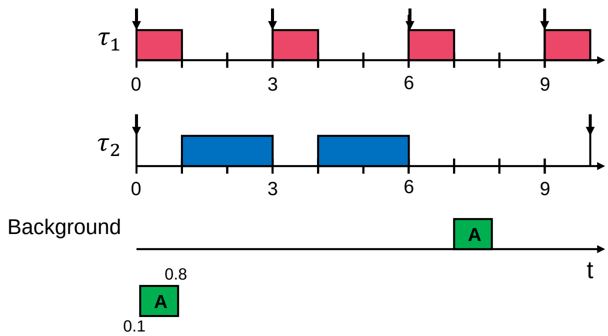
- Aperiodic tasks are only scheduled and executed at times, when there is no periodic task ready for execution, i.e. the processor is idle



14

Background Execution

$\tau_1 = (3,1)$, $\tau_2 = (10,4)$



Background Execution

Advantage: simple algorithm

Disadvantage: aperiodic jobs are often executed very late

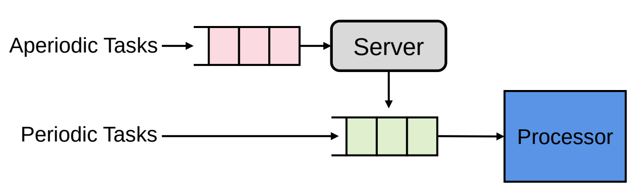
Periodic Server

- A task that behaves more or less like a periodic task and is created to execute aperiodic tasks is called a *periodic server*
- A periodic server is defined partially by execution time C_S and period T_S
- The parameter C_S is the execution capacity of the server (also called its budget)
- The ratio $U_S = C_S / T_S$ is the size of the server

A periodic server can generally be scheduled with the same algorithms as periodic tasks

Periodic Server

- When the server is scheduled it executes aperiodic tasks. While doing that, it *consumes* its budget at the rate of 1 per time unit
- The budget is *exhausted*, when it reaches 0
- A time instant when the budget is replenished (reloaded) is called *replenishment time*



Periodic Server

- A periodic server is *backlogged* whenever the aperiodic task queue is non-empty
- It is *idle* when the queue is empty
- The server is *eligible* for execution only when it is backlogged and has non-zero budget

Polling Server (PS)

- A *polling server* (T_s, C_s) is a periodic server
- When executed, it executes an aperiodic task, if the aperiodic task queue is non-empty
- PS suspends execution or is suspended by the scheduler either
  - when it has executed for C_s, or
  - when there is no aperiodic request pending

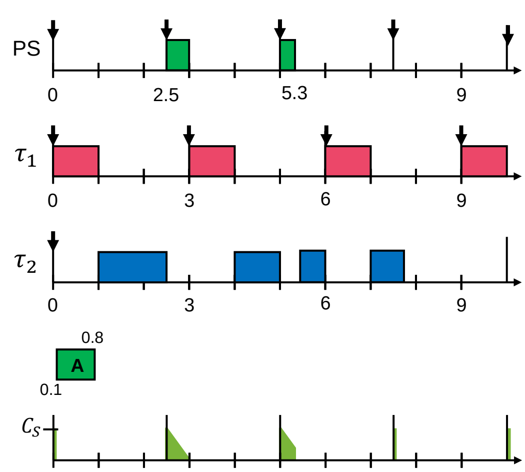
Polling Server

- Consumption Rules
  - The budget is immediately consumed when the server is not scheduled[1]
- Replenishment Rules
  - The budget is replenished to C_s at the beginning of each period
- Example: Liu, Figure 7.2b, p.193

[1]If a request arrives just after the server suspended it has to wait for the next period!

21

Polling Server τ_1 = (3,1), τ_2 = (10,4), PS =(2.5, 0.5)



22

Polling Server

- Aperiodic jobs that arrive after the release time of the PS must wait until next polling period
  – Execution budget is not preserved
- Simple to prove correctness

Polling Server – Schedulability Analysis

- The schedulability of periodic tasks must be guaranteed in the presence of the polling server.
- From the definition of the PS we can see that
  - In the worst case the server behaves like a periodic task where $\mathrm{T} = T_S$ and $\mathrm{C} = C_S$

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_S}{T_S} \leq (n+1) \cdot \left(2^{1/(n+1)} - 1\right)$$

Polling Server

Bandwidth-Preserving Servers

- A bandwidth-preserving server is a periodic server
- Compared to polling server bandwidth preserving servers try to preserve their budget when they are not executed
- Additional rules for consumption and replenishment

Bandwidth-Preserving Servers

- A backlogged bandwidth-preserving server is ready for execution when it has budget
- Scheduler keeps track of the consumption of the server budget
- If budget is exhausted server becomes idle

Bandwidth-Preserving Servers

- Scheduler moves server back to ready queue, when budget is replenished and server is backlogged
- If a new aperiodic job arrives an idle server becomes backlogged and is put into the ready queue when it has budget

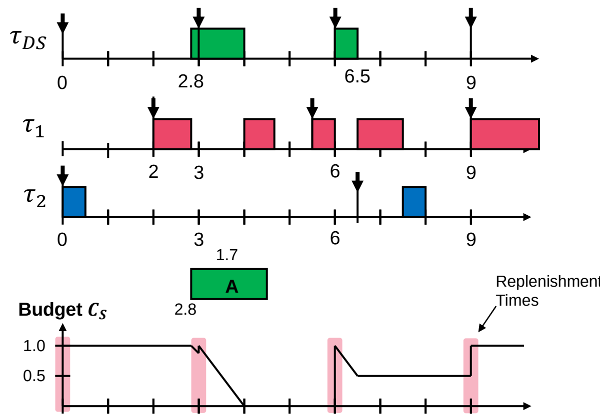
Deferrable Server (DS)

- Simplest bandwidth preserving server
- Consumption rule
  - The execution budget of the server is consumed at the rate of one unit per time whenever the server executes
- Replenishment rule
  - The execution budget of the server is set to C_s at multiples of its period
- Server is not allowed to cumulate budget from period to period

Introduced to improve the average response time compared to the Polling Server

Deferrable Server (RMS)

τ_{DS} = (3,1), τ_1 = (2, 3.5, 1.5, 3.5), τ_2 =(6.5, 0.5)



Deferrable Server – Schedulability Analysis

- All schedulability tests and timing analysis methods related to RM had the assumption:

A periodic task cannot suspend itself

- Since the DS preserves its capactity it violates this assumption!

Schedulability of Deferrable Servers

- Time Demand Analysis can be used to determine whether all tasks remain schedulable in the presence of a deferrable server
- Time Demand Function (if deferrable server has highest priority)

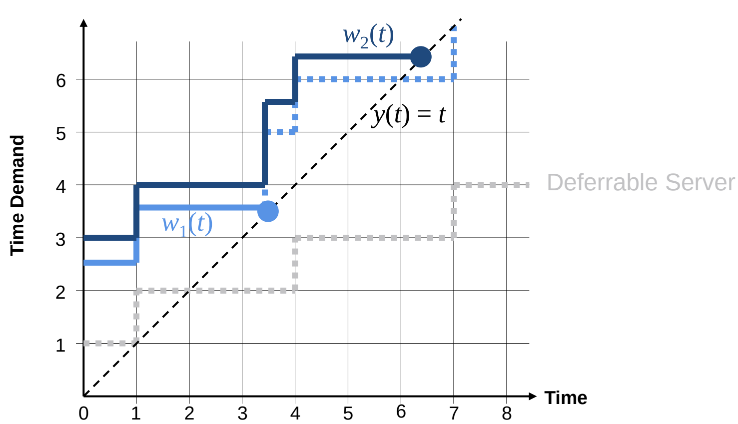
$$w_i(t) = C_i + B_i + C_S + \left\lceil \frac{t - C_S}{T_S} \right\rceil \cdot C_S + \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j \quad ,\forall 0 \le t \le T_i$$

Blocking

Server

Deferrable Server
Time Demand Analysis

τ_{DS} = (3,1), τ_1 = (2,3.5,1.5), τ_2 =(6.5, 0.5)



Schedulability of Deferrable Servers

- There is no known schedulability utilization that ensures the schedulability of a fixed-priority system in which a deferrable server is scheduled at an arbitrary priority

Limitations of Deferrable Servers

- Deferrable servers may be scheduled longer than its execution time in a time interval as long as its period
- Lower priority task may be delayed longer by a deferrable server than by a periodic task with same period and execution time

Deferrable server does not behave as a periodic task

Sporadic Server (SS)

- Bandwidth preserving server
- Improves the average response time of aperiodic tasks without degrading the utilization bound for periodic tasks
- More complex consumption and replenishment rules ensure that each sporadic server with period T_s and budget C_s never demands more processor time than the periodic task (T_s, C_s) in *any* time interval

Sporadic Server – Additional Terms

- The priority of a task that currently executes is π_{exe}
- The SS has the associated priority π_S
- If $\pi_{exe} \geq \pi_S$ the SS is Active
- If $\pi_{exe} < \pi_S$ the SS is Idle
- The Replenishment Time of the SS is **RT**
- The Replenishment Amount of the SS is **RA**

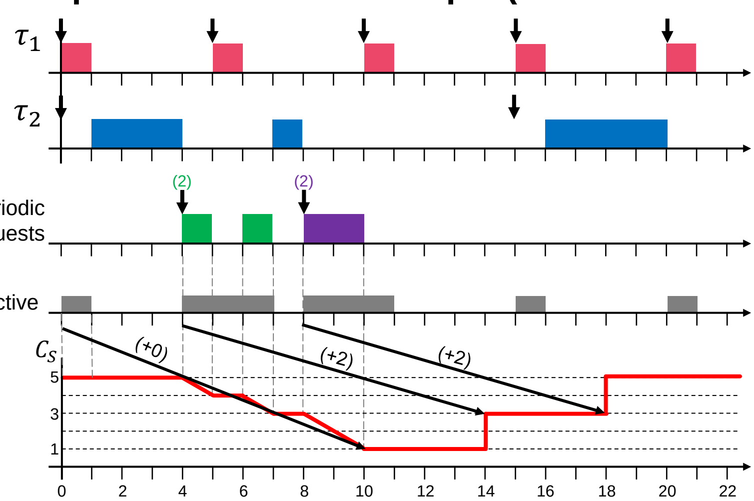
Sporadic Server – Replenishment Rules

The Sporadic Server replenishes its capacity based on the following rules:

1. RT is set as soon as SS becomes active and $C_S > 0$
   - Example: SS becomes active at time t_A and $C_S > 0$, then RT= $t_A + T_S$
2. For a certain RT, RA is computed when SS becomes idle or when the capacity is exhausted ($C_S = 0$)
   - Example: The SS becomes idle at t_I. Then RA is set equal to the capacity that has been consumed in the interval $[t_A, t_I]$

Sporadic Server – Example (Buttazzo P.145)

$C_S = 5$
$T_S = 10$



Sporadic Server – Schedulability Analysis

Theorem (Sprunt, Sha, Lehoczky): (Buttazzo p. 146)

A periodic task set that is schedulable with a task τ_i is also schedulable if τ_i is replaced by a Sporadic Server with the same period and execution time.

→ The same schedulability test that was used for the Polling Server can also be used for the Sporadic Server.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_S}{T_S} \leq (n+1) \cdot \left(2^{1/(n+1)} - 1\right)$$

Sporadic Server

Fixed-Priority Sporadic Server

- Sporadic server is more complex than polling or deferrable servers due to more complex consumption and replenishment rules
- Main advantage: schedulability easy to demonstrate
- A sporadic server can be treated like a periodic task when we check for schedulability
- System with sporadic server may be schedulable while the corresponding Deferrable Server is not
- More complex sporadic servers exist

Sporadic Dynamic-Priority Servers

- Sporadic servers can also be used for dynamic-priority (deadline-driven) systems, like EDF
- Rules have to be adapted
- Also, here sporadic server can be treated as normal task for schedulability analysis

Other Bandwidth Preserving Servers

- Other bandwidth preserver algorithms are based on *general processor sharing (GPS)* algorithms (deadline-driven algorithms)
- Examples:
  - Constant utilization server
  - Total bandwidth server
  - Weighted round-robin server
- Exact functionality not discussed in course, instead see Buttazzo: 6

Outline

- System Model and Assumptions
- Scheduling Aperiodic Tasks (with soft deadlines)
- Scheduling Aperiodic Tasks (with firm deadlines)

Scheduling Aperiodic Jobs with Firm Deadline

- Aperiodic tasks with firm deadlines
  - Scheduler decides, if a new job request can be accepted or must be rejected
    - Job is accepted and scheduled, if all other scheduled job still meet their deadlines
    - Otherwise, job is rejected
- Aperiodic job with firm deadlines is denoted by $S_i(a_i, d_i, C_i)$

Acceptance Test in Fixed-Priority System

- Sporadic server can be used to execute sporadic tasks in a fixed-priority system
- The sporadic server (T_s, C_s) has C_s units of processor time every T_s units of time

45

Acceptance Test in Fixed-Priority Systems

- For each new job $S(a, d, C)$ of an aperiodic job it must be checked, if
  - new job can be scheduled together with the aperiodic jobs with firm deadline that have deadline before d
  - aperiodic jobs with firm deadline with deadline larger or equal to d can still be scheduled
- Acceptance test is quite complex, but may still be feasible for many systems

46

Acceptance Test in Fixed-Priority Systems

- Accepted jobs of aperiodic tasks with firm deadlines are ordered among themselves on EDF basis
- For the first aperiodic job with firm deadline $S_1(t, d_{s,1}, C_{s,1})$ the server has at least $\left\lfloor \frac{d_{s,1}-t}{T_s} \right\rfloor \cdot C_s$ units of processor time available
- Thus, first job is accepted, if the slack of the task is larger than or equal to 0.

$$\sigma_{s,1}(t) = \left\lfloor \frac{d_{s,1}-t}{T_s} \right\rfloor \cdot C_s - C_{s,1}$$

Acceptance Test in Fixed-Priority Systems

- When there are already n accepted aperiodic jobs with a firm deadline in the system, the scheduler computes the slack $\sigma_{s,i}$ of S_i according to

$$\sigma_{s,i}(t) = \left\lfloor \frac{d_{s,i}-t}{T_s} \right\rfloor \cdot C_s - C_{s,i} - \sum_{d_{s,k}<d_{s,i}} (C_{s,k} - \xi_{s,k})$$

where $\xi_{s,k}$ is the execution time of the completed portion of the aperiodic job with firm deadline S_k

(ξ = xi)

Acceptance Test in Fixed-Priority Systems

- If the slack $\sigma_{s,i}$ for the aperiodic job with firm deadline is not less than 0, we have to check, if all accepted jobs can still meet their deadline
- For each aperiodic job with firm deadline S_k which has an equal or later deadline than S_i, we have to check, if the slack $\sigma_{s,k}$ is larger than the execution time of the new aperiodic job $C_{s,i}$
- The new aperiodic job with firm deadline is only accepted, if this is the case for all accepted aperiodic jobs with firm deadline

Summary

- Servers can be used for the efficient scheduling of aperiodic jobs with soft and firm deadlines
- Servers have consumption and replenishment rules, which can be arbitrarily complex
- Implementation overhead can be significant

References

[1] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic respon- siveness in hard real-time environments. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1987.

[2] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algo- rithm for enhancing aperiodic responsiveness in hard-real-time environ- ments. *IEEE Transactions on Computers*, 4(1), January 1995.

[3] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. Journal of Real-Time Systems, 1(1):27–60, 1989.

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

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