Scheduling Aperiodic and Sporadic Jobs in Priority-Driven Systems

Ingo Sander
ingo@kth.se

Liu: Chapter 7

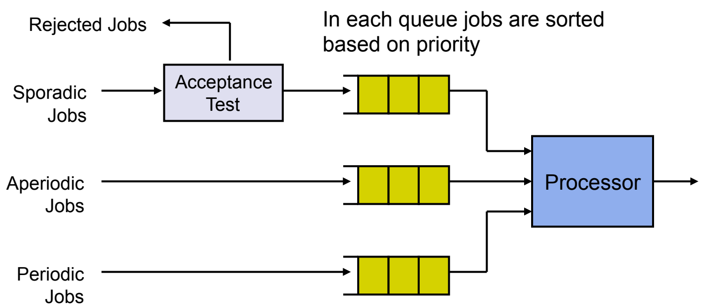
Outline

- System Model and Assumptions
- Scheduling Aperiodic Jobs
- Scheduling Sporadic Jobs

Assumptions

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

System Model



Scheduling Algorithms

- Aperiodic jobs
  - are always accepted
  - Scheduler tries to complete aperiodic jobs as soon as possible

Scheduling Algorithms

- Sporadic jobs
  - 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

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 sporadic tasks always meet their deadlines

Optimality of Algorithms

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

Optimality of Algorithms

- A sporadic job scheduling algorithm is optimal, if it
  - accepts each sporadic job newly offered to the system and schedules the job to complete in time *if and only if* the new job can be correctly scheduled in time by some means

Scheduling Aperiodic Jobs

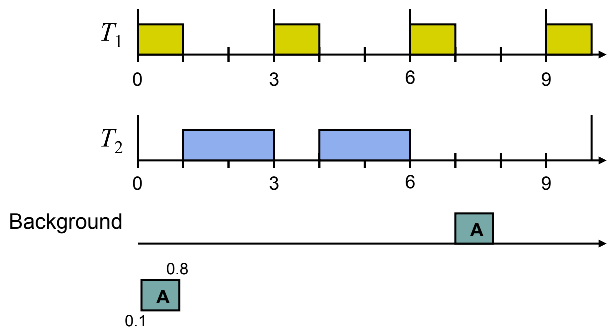
- In the following several different algorithms to schedule aperiodic jobs are discussed
- Assumption here:
  - No sporadic jobs

Background Execution

- Aperiodic jobs are only scheduled and executed at times, when there is no periodic or sporadic job ready for execution

Background Execution

$T_1 = (3,1)$, $T_2 = (10,4)$



Background Execution

- Advantage: simple algorithm
- Disadvantage: aperiodic jobs are often executed very late
- Possible improvement: *slack stealing* (as shown in chapter 5 for clock-driven systems)
  - slack stealing can greatly improve the performance, but is much more complex in priority-driven systems

Periodic Server

- A task that behaves more or less like a periodic task and is created to execute aperiodic jobs is called a *periodic server*
- A periodic server is defined partially by execution time e_S and period p_S
- The parameter e_S is the execution budget of the server
- The ratio $u_S = e_S / p_S$ is the size of the server

Periodic Server

- When the server is scheduled and executes aperiodic jobs, 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 job 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

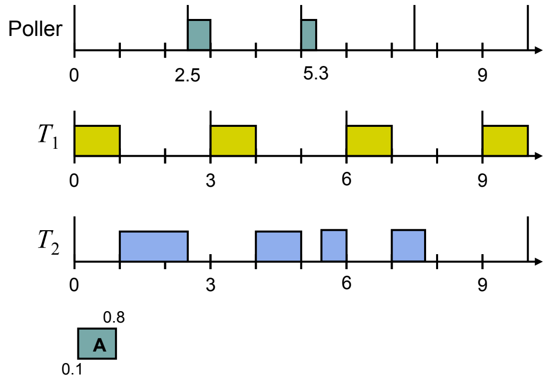
- A *polling server* (p_s, e_s) is a periodic server
- When executed, it executes an aperiodic job, if the aperiodic job queue is non-empty
- Poller suspends execution or is suspended by the scheduler either
  - when it has executed for e_s, or
  - when the aperiodic job queue becomes empty

Polling Server

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

Polling Server

$T_1 = (3,1)$, $T_2 = (10,4)$, Poller =(2.5, 0.5)



Polling Server

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

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

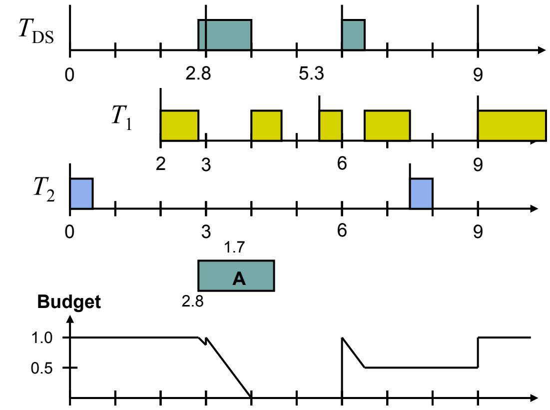
- Scheduler moves server back to ready queue, when budget is replenished and server is backlogged
- If a new aperiodic job arrives an idle server becomes backlogged and is put into the ready queue when it has budget

Deferrable Server

- 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 e_s at multiples of its period
- Server is not allowed to cumulate budget from period to period

Deferrable Server (RMS)

T_{DS} = (3,1), T_1 = (2,3.5,1.5), T_2 =(6.5, 0.5)



Schedulability of Deferrable Servers

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

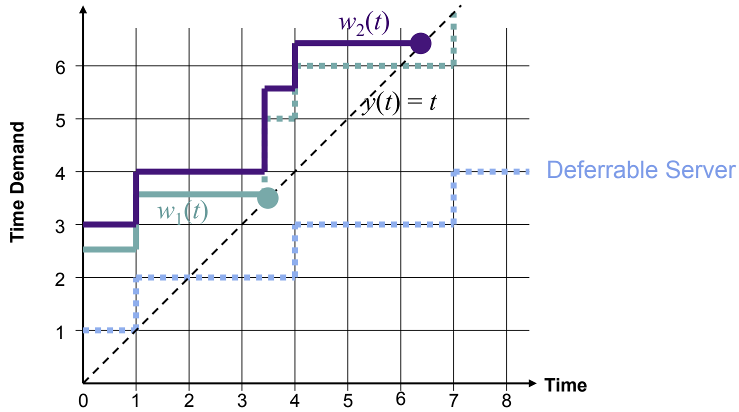
$$w_i(t) = e_i + b_i + e_s + \left\lceil \frac{t - e_s}{p_s} \right\rceil e_s + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k, \quad \text{for } 0 \le t \le p_i$$

Blocking

Server

Deferrable Server Time Demand Analysis

T_{DS} = (3,1), T_1 = (2,3.5,1.5), T_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
- Some special cases are discussed in the book (Liu p.201)

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

- Bandwidth preserving server
- More complex consumption and replenishment rules ensure that each sporadic server with period p_s and budget e_s never demands more processor time than the periodic task (p_s, e_s) in *any* time interval

A simple Fixed-Priority Sporadic Server

- Definitions (Liu p.205)
  - t_r denotes the latest actual replenishment time
  - t_f denotes the first instant after t_r at which the server begins to execute
  - t_e denotes the latest *effective replenishment time*

A simple Fixed-Priority Sporadic Server

- Definitions (Liu p.205)
  - At any time t, $BEGIN$ is the beginning instant of the earliest busy interval among the latest contiguous sequence of busy intervals of the higher priority subsystem $\mathbf{T}_H$ that started before t
  - END is the end of the latest busy interval in the above defined sequence if this interval ends before t and equal to its infinity if the interval ends after t
- *Server Busy Interval:* Begins when an aperiodic job arrives at an empty aperiodic job queue and ends when the queue becomes empty again

Fixed-Priority Simple Sporadic Server

- Consumption Rules (Liu p.206)
  - At any time t after t_r, the server's execution budget is consumed at the rate of 1 per unit time until the budget is exhausted when either one of the following to conditions are true. When these conditions are not true, the server holds its budget
    - **C1** The server is executing
    - **C2** The server has executed since t_r and $END < t$

Fixed-Priority Simple Sporadic Server

- Replenishment Rules (Liu p.206)
  - **R1** Initially when the system begins execution and each time when the budget is replenished, the execution budget = e_s, and t_r = the current time
  - **R2** At time t_f, if $END = t_f$, $t_e = \max(t_r, BEGIN)$. If $END < t_f$, $t_e = t_f$. The next replenishment time is set at $t_e + p_s$

Fixed-Priority Simple Sporadic Server

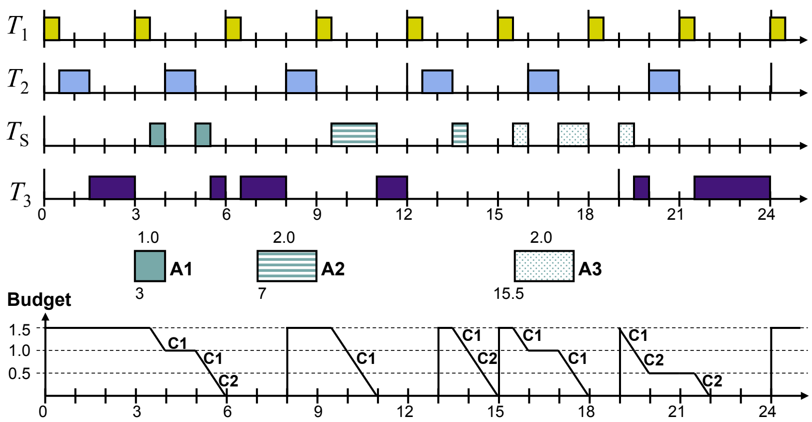
- Replenishment Rules (Liu p.206)
  - **R3** The next replenishment occurs at the next replenishment time, except under the following conditions. Under these conditions, replenishment is done at times stated below
    - a) If the next replenishment time $t_e + p_s$ is earlier than t_f, the budget is replenished as soon as it is exhausted
    - b) If the system **T** becomes idle before the next replenishment time $t_e + p_s$ and becomes busy again at t_b, the budget is replenished at $\min(t_e + p_s, t_b)$

Simple Fixed-Priority Sporadic Server

- Example: Liu, Figure 7-8, p.207

Simple Fixed-Priority Sporadic Server - Example

$T_1 = (3{,}0.5)$, $T_2 = (4{,}1)$, $T_3 = (19{,}4.5)$ $T_S = (5,\ 1.5)$



Simple Fixed-Priority Sporadic Server - Example

t	t_r	t_f	t_e	*BEGIN*	*END*	
3.5	0	3.5	3	3	3.5	**R2**: $t_e = \max(0,3.0)$ => $t_r^+ = t_e + t_s = 8.0$
3.5 5.0						**C1:** Server executes **C2:** Queue Empty, higher priority jobs idle
8						**R3:** Replenishment
9.5	8	9.5	3	8	9.5	**R2**: $t_e = \max(8,8)$ => $t_r^+ = t_e + t_s = 13.0$

Simple Fixed-Priority Sporadic Server - Example

t	t_r	t_f	t_e	$BEGIN$	END	
13.5	13	13.5	13	12	13.5	**R2**: $t_e = \max(13,12)$ => $t_r^+ = t_e + t_s = 18.0$
13.5						System T idle! **R3b**: $t_r^+ = \min(t_e + p_s, t_b) = 15.0$, since system ($T_1$) becomes busy at 15
15.5	15	15.5		15	15.5	**R2**: $t_e = \max(15,15)$ => $t_r^+ = t_e + t_s = 20.0$
18.5						System T idle! **R3b**: $t_r^+ = \min(t_e + p_s, t_b) = 19.0$, since system ($T_3$) becomes busy at 19

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 (Liu: 7.3.2)

Sporadic Dynamic-Priority Servers

- Sporadic servers can also be used for dynamic-priority (deadline-driven) systems
- Rules have to be adapted to EDF or LST
- 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 Liu: 7.4

Scheduling Sporadic Jobs

- Sporadic jobs
  - 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 job is denoted by $S_i(r_i, d_i, e_i)$

Acceptance Test in Fixed-Priority System

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

Acceptance Test in Fixed-Priority Systems

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

Acceptance Test in Fixed-Priority Systems

- Accepted sporadic jobs are ordered among themselves on EDF basis
- For the first sporadic job $S_1(t, d_{s,1}, e_{s,1})$ the server has at least $floor((d_{s,1} - t)/p_s)\ e_s$ units of processor time available
- Thus first job is accepted, if the slack of the job

  $$\sigma_{s,1}(t) = floor((d_{s,1} - t)/p_s)\ e_s - e_{s,1}$$

  is larger than or equal to 0.

Acceptance Test in Fixed-Priority Systems

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

$$\sigma_{s,i}(t) = \text{floor}((d_{s,i} - t)/p_s)e_s - e_{s,i} - \sum_{d_{s,k} < d_{s,i}} (e_{s,k} - \xi_{s,k})$$

where $\xi_{s,k}$ is the execution time of the completed portion of the sporadic job S_k

Acceptance Test in Fixed-Priority Systems

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

Summary

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