AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Architecture & Design of Embedded Real-Time Systems (TI-AREM)

Handling of Asynchronous Events - Sporadic server algorithms

Version: 5-3-2015

Agenda

- What is the problem?
- 4 Methods of handling asynchronous events
  - Use an Interrupt handler
  - Service the event at a specified SW priority
  - Use a Polling task
  - Use a Sporadic Server
- The Sporadic Server algorithm

What is the problem ?

- Rate Monotonic Analysis assumes **periodic signals**
- How can we incorporate aperiodic or asynchronous events in this analysis?
- How do we handle aperiodic or asynchronous events or signals?
- Can we handle both **bounded** arrival and **unbounded** arrival of aperiodic events?

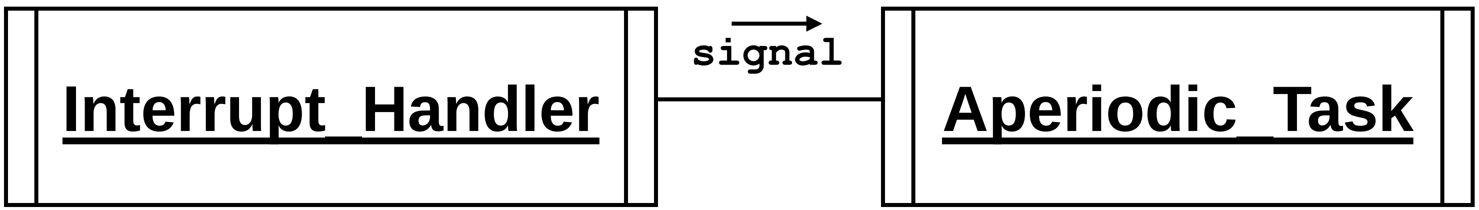
M1: Use an Interrupt Handler

```
Interrupt_Handler is
begin
    read event input
    process aperiodic event
    reset interrupt HW for next event
end Interrupt_Handler
```

Interrupt handlers executes at the highest system priority

1. **Only usable for very short event response time**
2. **May be dangerous if arrival bound is exceeded**

M2: Service the event at a specified SW priority (1)




```
Interrupt_Handler is
begin
    read event input
    Signal Aperiodic_Task
    reset interrupt HW for next event
end Interrupt_Handler
```

M2: Service the event at a specified SW priority (2)

```
Aperiodic_Task is
begin
    loop
        wait for Interrupt_Handler signal
        process aperiodic event
    end loop
end Aperiodic_Task
```

**NB! Task priority assigned according to deadline monotonic assignment
(shortest deadline => highest priority**

M3: Use a Polling Task (1)

- Polling is a well-known and very predictable approach for scheduling aperiodic events
- Must be used when events do not cause interrupts
- Hard deadlines that are short relative to the minimum interarrival time requires short polling periods
- Overhead may be excessive when timing requirements are short

M3: Use a Polling Task (2)

```
Polling_Task is
begin
  loop
    if Event_Arrived then
       Reset Event_Arrived flag or HW
       Process aperiodic event
    end if
    next_start= next_start + polling_period
    Sleep_Until next_start
  end loop
end Polling_Task
```

NB! The worst case response time occurs when the event arrives just after the polling task has checked

M4: Use a Sporadic Server

- The sporadic server is a mechanism for scheduling aperiodic activities in time-critical systems
- The **sporadic server algorithm** [Sprunt89]
- A sporadic server preservers and limits a certain amount of **execution capacity** for processing of aperiodic events

[Sprunt89]: *"Aperiodic Task Scheduling for Hard Real-Time Systems"* B. Sprunt et. Al. 1989

Polling task versus Sporadic Server

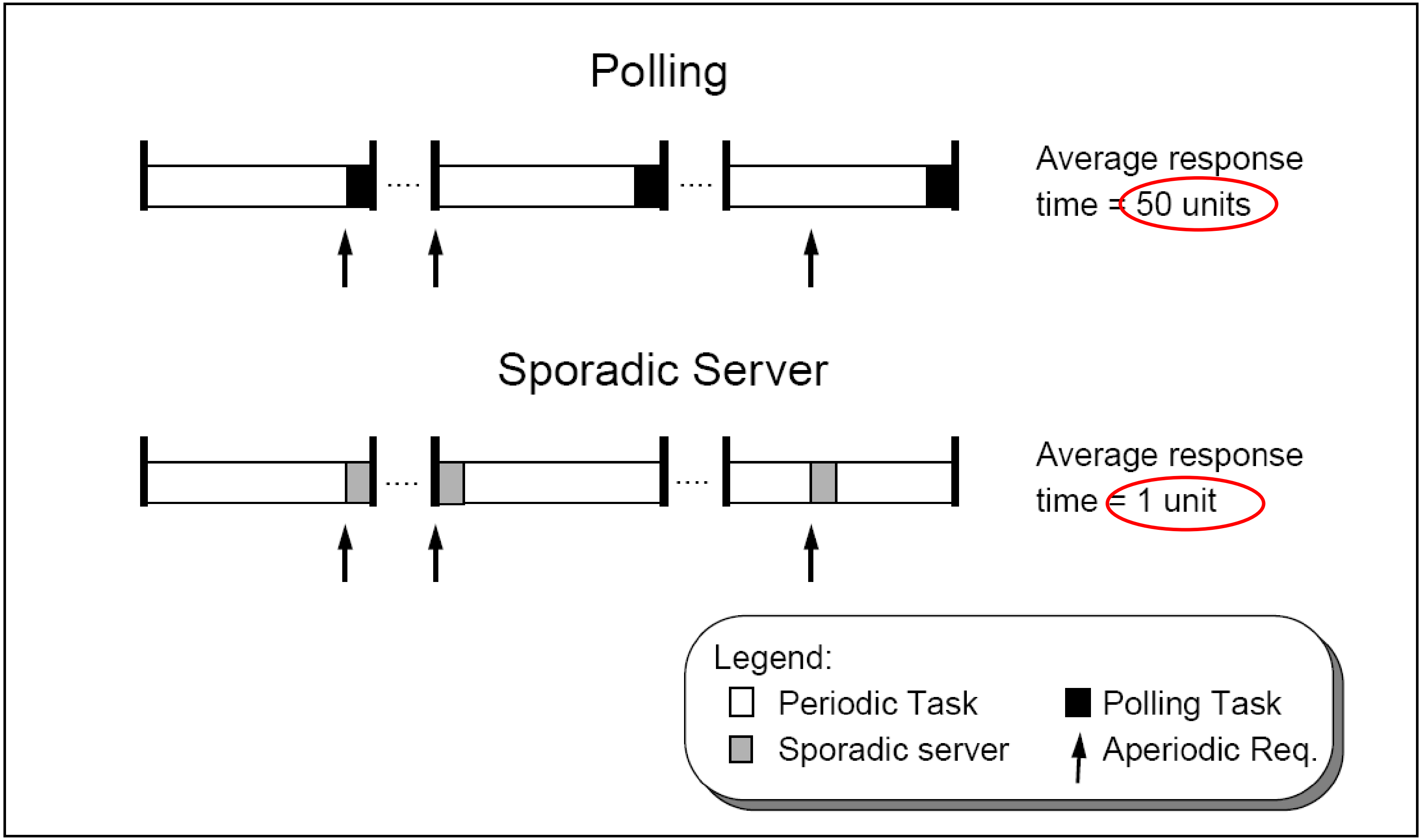


Figure 1 Comparison Between a Sporadic Server and a Polling Task

Sporadic Server

- The sporadic server is event-driven from an application viewpoint
- **Appears as a periodic task** for the purpose of analysis
  - Allows the use of e.g. Rate Monotonic Analysis to predict the behavior of real-time systems

The Sporadic Server Algorithm

The **sporadic server algorithm** is characterized by three parameters:

1. **Priority**
   - at which the **response** to the aperiodic event executes
2. **Execution capacity**
   - assigned to the aperiodic tasks
   - **When the capacity is exhausted**, the aperiodic task must either **relinquish** the processor or **execute** at a **background priority**
3. **Replenishment period**
   - a duration of time that must pass before the exhausted execution capacity is **restored**

Schedulability of Sporadic Servers

- A set of ***n*** independent periodic tasks scheduled by the ***Rate Monotonic Algorithm*** will always meet its deadlines, for all task phasing's, if

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \le n(2^{1/n} - 1) = U(n)$$

where

C_i = worst-case task execution time of task_i

T_i = period of task_i

U(n) = utilization bound for *n* tasks

For a sporadic server task use:

$$\frac{C_i}{T_i} = \frac{\textbf{Execution Capacity Time}}{\textbf{Replenishment period}}$$

Sporadic Server

- Two Implementation solutions:
  - Run-time support in RTOS
    - Sporadic server policy included in the real-time extension (IEEE1003.1d) to the POSIX standard in 1999
  - Application level implementation
    - see the CMU Report SEI-91-TR-26

Sporadic Server with RTOS Runtime Support

```
Interrupt_Handler is
begin
  read event input
  signal Aperiodic_Task
End Interrupt_Handler

Aperiodic_Task is
begin
  Initialize_Sporadic_Server(
       Replenish_Period(=Min_Interarival_Int),
       Execution_Capacity(=Worst_Case_Exec_time))
  loop
    wait for interrupt signal
    process aperiodic event
  end loop
end Aperiodic_Task
```

Sporadic Server: SW Implementation

```
Interrupt_Handler is
begin
  read event input
  signal Aperiodic_Task
  Interrupt_Time:= ReadClock
End Interrupt_Handler

Aperiodic_Task is
begin
  loop
    wait for interrupt signal
    process aperiodic event
    Sleep_Until Interrupt_Time + Replenish_Period
  end loop
end Aperiodic_Task
```

NB! Simplified version: Execution budget = 1 event handling

Sporadic Server Task Example

Execution capacity= 10 ms
Replenishment period= 18 ms, WCE= 5 ms

High

Medium

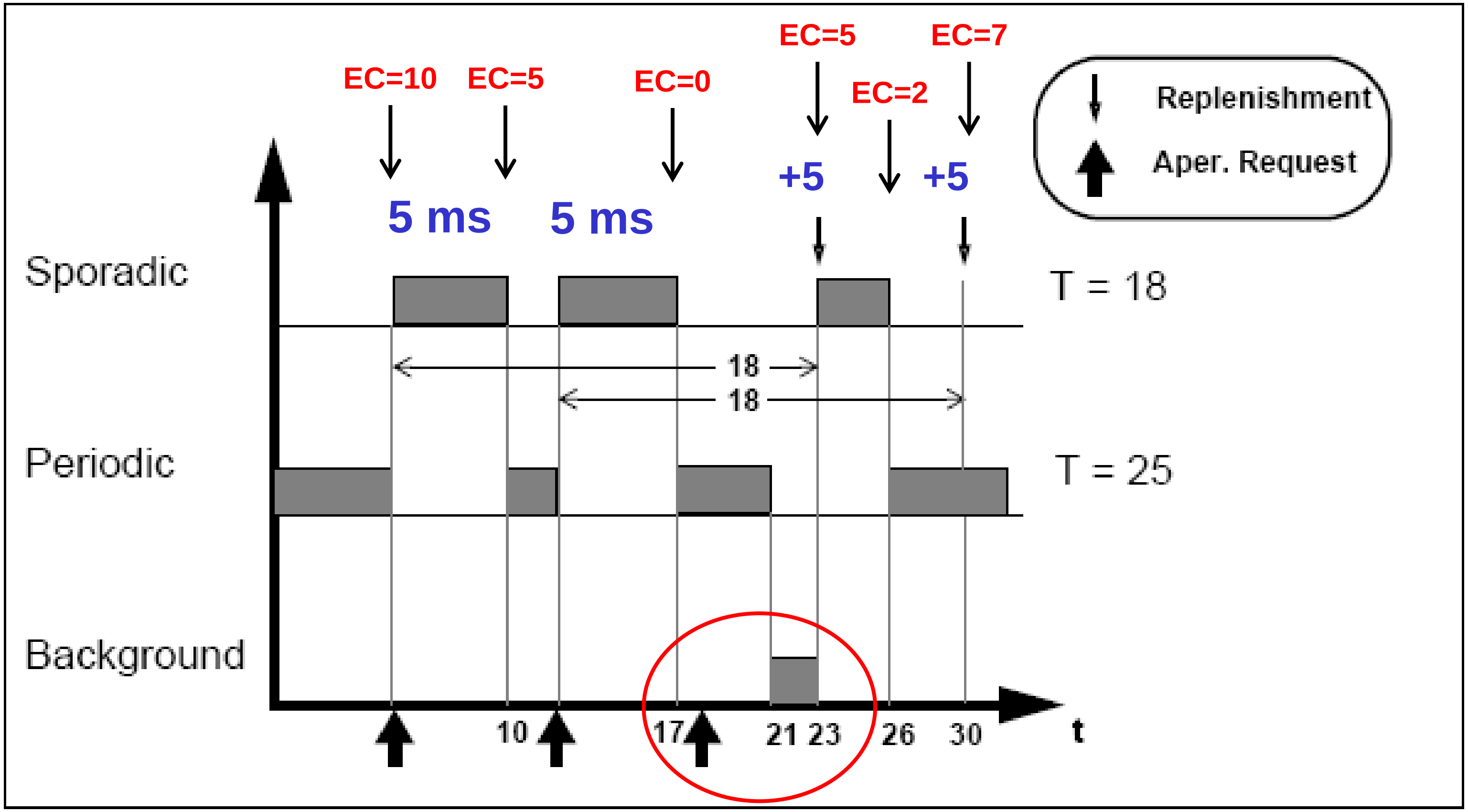


Figure 2 Example of a Sporadic Server-Controlled Task

Sporadic Server:
An Application Level Implementation

- Based on document: CMU/SEI-91-TR26
  - *”An application-level implementation of the sporadic server”*
  - by Michael Gonzáles and Lui Sha, Software Engineering Institute, Carnegie Mellon University, 1991.
  - Contains an Ada Task Implementation and a Library-level sporadic server implementation based on the **POSIX standard**

Algorithm Assumption in TR26

1. Measurement of execution time
   - Instead of the actual execution time, **the worst-case execution time** (**WCE**) of each request is used for budget consumption
   - The aperiodic task cannot be initiated unless there is **enough available execution time**
2. Replenishment Policy
   - An intermediate replenishment policy is used
   - If an aperiodic event arrives at time **t** and an execution budget **Q** is available, this budget has to be replenished at time **t+T** (where T is the replenishment period)
3. Background processing
   - Using dynamic priority mechanism, the application-level sporadic server can change the priority from normal to background

Sporadic Server Example

Periodic task: **6 ms period, WCE= 3 ms**
Sporadic server: **Replenishment time= 8 ms, WCE 2 ms**
Execution Capacity= 2 ms

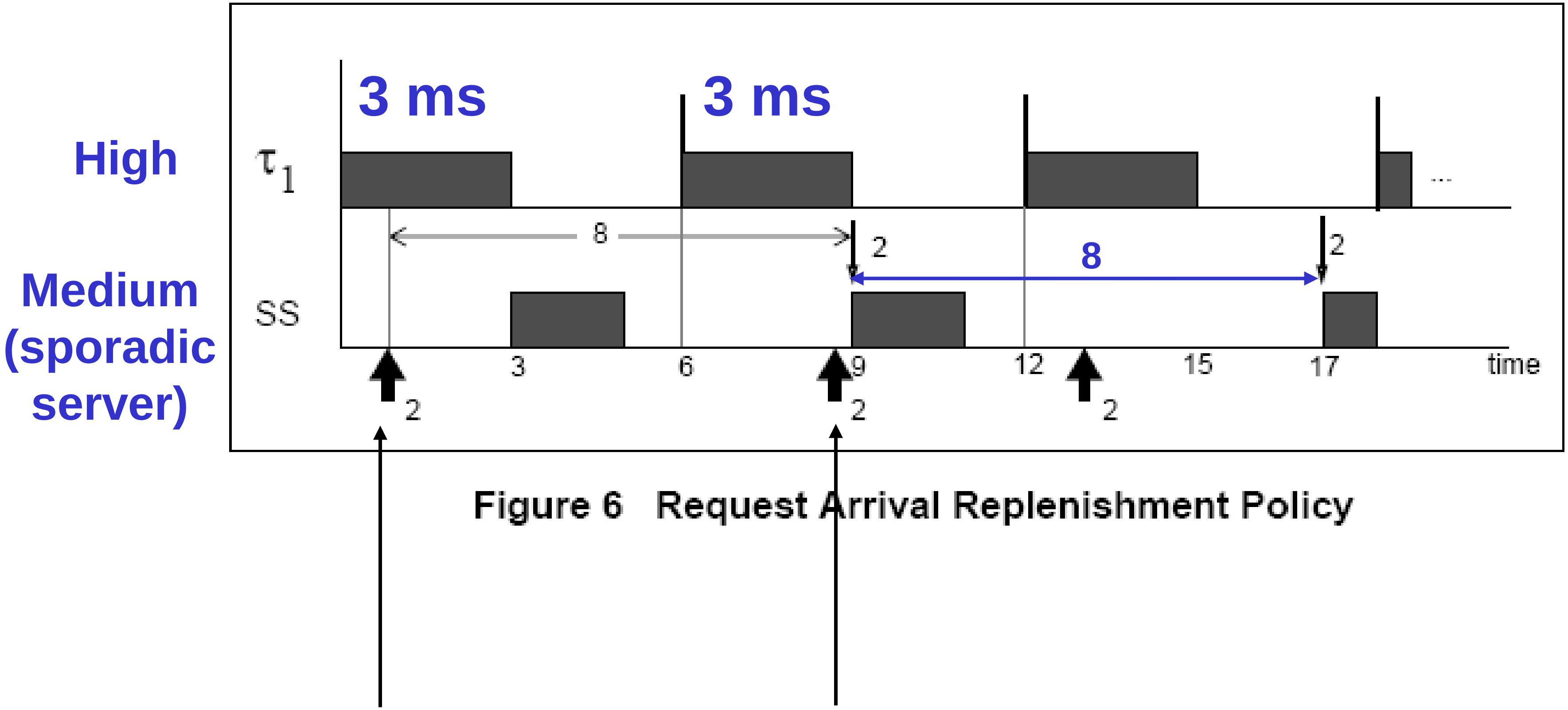


Figure 6 Request Arrival Replenishment Policy

Algorithm Implementation (TR26)

- The aperiodic task is forced to wait at the **highest priority** in the system
- The sporadic server is activated precisely at the instant of the arrival of the aperiodic request
- As soon as the aperiodic request arrives the aperiodic task is activated and the replenishment is scheduled.

```
If sufficient execution capacity is
available then
    the priority is set to its normal
    priority
 else
     it is lowered to background priority
```

POSIX Sporadic Server Interface

- pthread_ss_init()
  - Initalize a sporadic server to control the calling thread
- pthread_ss_arm()
  - Arm the sporadic server to wait for an aperiodic request
- pthread_ss_request()
  - Initiate the processing of an aperiodic request
- pthread_ss_detach()
  - Detach all references to the sporadic server

User Code for an Aperiodic Thread

```
#include <sys/timers.h>
#include <pthread.h>

void run()          // aperiodic thread
{
   Declare the sporadic server variables sserver;

   pthread_ss_init(&sserver, replinishment_period, execution_budget,
                   normal_priority, background_priority);     // POSIX call
   while (1)
   {
      pthread_ss_arm(&sserver);                               // POSIX call
      wait for an aperiodic event;
      pthread_ss_request(&sserver, WCE_time);                 // POSIX call
      process the aperiodic event;
   }
}
```

Summary

- Sporadic server algorithm an alternative to traditional techniques
- An effective algorithm compared to polling
- Allows RMA schedulability analysis for aperiodic tasks
- Supported either by RTOS or by application level programming