Supporting Component-Based Development with Hierarchical Scheduling

Introduction
- Partitioning into multiple simpler subsystems
  - Lower complexity;
  - Component reuse;
  - Team-base development;
  - Outsourcing.

Integration
- Without reservations

Automotive Example: Engine Control + ABS

Hierarchical Scheduling Framework

Integration
**Introduction**

**Automotive Example**: Engine Control + ABS

Integration with reservations

Predictable interferences

**Shared Resources**

Tasks are usually **not independent**: they share resources!

Examples:
- Data Structures, Peripheral Devices, Common Memory Areas

**HSF**

- Scheduling mechanisms needed to implement the Hierarchical Scheduling Framework (HSF)
  - Resource reservation server able to guarantee hard real-time applications;
  - Resource sharing protocol supporting resources shared among tasks running upon different reservation servers.

**Resource Reservation**

**HARD reservation**

It guarantees that the served application receives **at most** a budget Q every period P.

**Problems with Reservations**

- Resource sharing may break isolation:

  - deadline miss
  - normal blocking due to resource sharing
  - extra blocking due to budget exhaustion

The budget is recharged at the server deadline

Hard-CBS Server

<table>
<thead>
<tr>
<th>Task</th>
<th>CPU Time</th>
<th>Server Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>T3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Server 1 (EDF)

Server 2 (FPPS)

Server n (EDF)
Problems with Reservations

- Resource sharing may break isolation:
  
  The major problem is that the resource is locked but no task is actually using it.

Overrun W/ Payback

A possible solution: When the budget exhausts inside a critical section, do nothing. Payback at the next budget replenishment.

Note that the worst-case bandwidth consumption does not change.

BROE

Check and recharge

If \( q_s \geq \delta \) then enter, else recharge the budget at full value and proportionally postpone the server deadline.

Note that off-line we must guarantee that \( Q_s \geq \max\{\delta\} \).

BROE Design Goals

- Avoiding budget overruns;
- Ensuring bandwidth isolation (i.e., each server must consume no more than \( \alpha = \frac{q}{p} \) of the processor bandwidth);
- Guaranteeing a bounded-delay partition to the served tasks.

BROE: budget check

- Consider a task \( \tau_1 \) accessing a resource \( R_k \) having \( \delta_k = 2 \)

BROE: budget check

- Consider a task \( \tau_1 \) accessing a resource \( R_k \) having \( \delta_k = 2 \)

BROE avoids budget overruns by performing the budget check.
BROE: budget check

- Consider a task $\tau_1$ accessing a resource $R_k$ having $\delta_k = 2$.

Checking point $q(t) = 1 < \delta_k = 2$

BROE Design Goals

- Overcome the problem of budget depletion inside critical sections.
  - Avoiding budget overruns;
  - Ensuring bandwidth isolation (i.e., each server must consume no more than $\alpha = \frac{1}{P}$ of the processor bandwidth);
  - Guaranteeing a bounded-delay partition to the served tasks.

BROE: bandwidth guarantee

- When the budget is not enough to complete the critical section, BROE performs a full budget replenishment;
- To not violate the server bandwidth, the budget replenishment must be reflected in a proportional deadline postponement.

BROE: bandwidth guarantee

- The idea of proportional deadline comes from a property of EDF scheduling with implicit deadlines;
- Suppose $\tau_i$ be schedulable with bandwidth (utilization) $\alpha_i = 0.5$.

\[ \alpha_i + \frac{1}{Q} \leq 1 \]

Consider a task $\tau_1$ accessing a resource $R_k$ having $\delta_k = 3$. Task $\tau_1$ executes on a BROE server configured with $Q=5$ and $P=10$.

Task $\tau_1$ executes on a BROE server configured with $Q=5$ and $P=10$.
Consider a task $\tau_1$ accessing a resource $R_k$ having $d_k = 3$. Task $\tau_1$ executes on a BROE server configured with $Q=5$ and $P=10$.

BROE: bandwidth guarantee

- The server has executed 8 units in a window of 16 units. The bandwidth $\frac{8}{16} = 0.5$ has not been violated.

BROE Design Goals

- Overcome to the problem of budget depletion inside critical sections
- Avoiding budget overruns;
- Ensuring bandwidth isolation (i.e., each server must consume no more than $\alpha = \frac{0}{P}$ of the processor bandwidth);
- Guaranteeing a bounded-delay partition to the served tasks.

BROE: bounded-delay

- The budget replenishment and the corresponding deadline postponement can easily result in a violation of the worst-case delay $\Delta = 2(P - Q)$, if not properly handled.
Consider a BROE server with \( Q = 4 \) and \( P = 8 \)

\( \tau_1 \) accesses a resource having \( \delta = 2 \)

The worst-case delay \( \Delta = 2(P - Q) \) is violated!

This is only an example, in the worst-case the delay can be potentially unbounded!

How to solve this problem?

The idea is to avoid to let the server execute "too much earlier" with respect to its deadline, after a budget replenishment

To guarantee a bounded-delay of \( \Delta = 2(P - Q) \), BROE imposes an explicit server suspension
To guarantee a bounded-delay of $\Delta = 2(P - Q)$, BROE imposes an explicit server suspension. If the server is “not executing too earlier”, it is not possible to violate the worst-case delay $\Delta$.

Depending on the execution state, BROE decides to suspend the server or not.

### BROE Design Goals
Overcome to the problem of budget depletion inside critical sections:
- Avoiding budget overruns;
- Ensuring bandwidth isolation (i.e., each server must consume no more than $\alpha = \frac{Q}{P}$ of the processor bandwidth);
- Guaranteeing a bounded-delay partition to the served tasks.

### BROE Resource Access Policy
Consider a BROE server having period $P$ and budget $Q$. The current budget at time $t$ is denoted as $q(t)$. When a task wishes to access a resource $R_k$ at time $t$:
- If $q(t) \geq \delta_k$, then enter the critical section (there is enough budget);
- Else, compute a recharging time $t_r = d - \frac{q(t)}{\alpha}$
  - If $t < t_r$, the server is suspended until time $t_r$, the budget is replenished to $Q$ and the deadline is shifted to $d = t_r + P$;
  - Otherwise, the budget is immediately replenished to $Q$ and $d = t_r + P$.

### BROE: constraints
- The BROE resource access policy can work only with EDF scheduling due to the proportional deadline shift. The support for FP scheduling of the servers is currently an open problem;
- In order to perform the budget check, BROE requires the specification of a worst-case holding time for the shared resources;
- BROE is intrinsically designed for the worst-case: the budget check can cause a scheduling decision that could be unnecessary.

### BROE: recap
- The BROE server is a scheduling mechanism providing resource reservation including the support for shared resources:
  - Hard reservation implementing the Hard-CBS algorithm;
  - Resource access protocol that guarantees both bandwidth isolation and bounded-delay to the served application.
In general, the BROE budget check has to be performed using the Resource Holding Time (RHT) of a shared resource;

\[ \text{RHT} = \text{budget consumed from the lock of a resource until its unlock} \]

Interference from high-priority task has to be accounted in the budget consumed when a resource is locked

\[ \text{RHT} = \text{Critical Section WCET + Worst-case Interference} \]

The interference is caused by the task preemptions

If resources are accessed in a non-preemptive manner, the RHT is equal to the worst-case critical section length;

Trade-off: lower threshold for the budget check, but greater task blocking due to non-preemptive blocking

Implementation Issues

- **Goal**: Implementation of a two-level Hierarchical Scheduling Framework using the BROE algorithm.

BROE Server: Hard-CBS + resource access policy

- Local scheduler: can be either EDF and FP

EDF/FP Scheduler

Classical Resource Sharing

Multi-layer scheduling infrastructure

Ready queue structure

OS with tick: the kernel comes into operation periodically, even if there are no scheduling events to be handled;

OS tick-less: the kernel come into operation only when is needed, i.e., in correspondence of scheduling events.

Example: budget management for reservation

We look at tick-less RTOS implementation on small microcontrollers.

EDF scheduling implementation: need for a timing reference having both

- High-resolution;
- Long life-time (to handle absolute deadlines).

Require 64 bit data structure for time representation

Deal with 64 bit data structures in small microcontrollers imposes a significant overhead in the scheduler implementation.
Implementation Issues

- **Circular timer**: avoid an absolute timing reference. The notion of time is relative with respect to a free running timer.
- Let $T$ the lifetime of the free running timer.
- It is possible to handle temporal events having a maximum spread of $T/2$.

Consider two events $e_1$ and $e_2$.

Let $t(e_1)$ be the absolute time of an event, and $r(e_i)$ its relative representation by using the circular timer.

To compare two events having $|t(e_1) - t(e_2)| < T/2$:
- If $(r(e_1) - r(e_2)) > 0$ then $t(e_1) > t(e_2)$
- If $(r(e_1) - r(e_2)) < 0$ then $t(e_1) < t(e_2)$
- If $(r(e_1) - r(e_2)) = 0$ then $t(e_1) = t(e_2)$

**Warning**: a relative representation becomes inconsistent after $T/2$!

- **Inactive servers**: It is necessary to perform a periodic check of inconsistent deadlines;
- A special timer has to be reserved for that job.

The implementation of EDF requires 2 timers:
- Free running timer
- Periodic timer for deadline consistency

**Hard-CBS Server**: its implementation requires to manage two main operations
- Budget enforcement;
- Budget recharge.

**Budget enforcement**: when the server starts to execute at time $t$, set up an one-shot timer with the current budget $q(t)$.

- If a preemption occurs, the timer is reconfigured; otherwise, it will fire to notify a budget exhaustion.

**Budget recharge**: when a server exhaust its budget, it has to be suspended until its deadline, where the budget will be recharged.

- A deadline-ordered queue of suspended servers has to be provided. Another one-shot timer triggers the budget recharge event for the first server in the queue.
**Implementation Issues**

- **Budget recharge**: when a server exhausts its budget, it has to be suspended until its deadline, where the budget will be recharged.
- A deadline-ordered queue of suspended server has to be provided. Another one-shot timer triggers the budget recharge event for the first server in the queue.

![Diagram of suspended servers queue and timers](image)

**One-Shot Timer**
d=10

**Queue of suspended servers waiting for budget replenishment**

**Implementation Issues**

- **Hard-CBS Server**: its implementation requires to manage two main operations
  - Budget enforcement;
  - Budget recharge.

The implementation of the Hard CBS requires 2 timers:
- One-shot timer for budget enforcement
- One-shot timer for budget recharge

**Implementation Issues**

- **BROE server suspension**: can be implemented exploiting the budget recharge queue
  - "If $t < t_c$, the server is suspended until time $t_c$."

![Diagram of suspended servers queue and timers](image)

**One-Shot Timer**
d=10

**Queue of suspended servers waiting for budget replenishment**

**Implementation Issues**

- **BROE server suspension**: can be implemented exploiting the budget recharge queue
  - "If $t < t_c$, the server is suspended until time $t_c$."

![Diagram of suspended servers queue and timers](image)

**One-Shot Timer**
d=10

**Queue of suspended servers waiting for budget replenishment**

---

**Thank you!**

Alessandro Biondi
alessandro.biondi@sssup.it