IN4343 Real-Time Systems

Priority Servers
Handing Hybrid Task Sets

Periodic tasks
+
Aperiodic tasks
Aperiodic task handling

- Aperiodic tasks are typically activated by external events (given by interrupts).
- From one hand, we want to reduce the response times of aperiodic tasks.
- On the other hand, we don’t want to jeopardize schedulability of periodic tasks.
Handling Criticality

- Aperiodic tasks with **HARD** deadlines must be guaranteed under worst-case conditions.

- Off-line guarantee is only possible if we can bound interarrival times (**sporadic tasks**).

- Hence **sporadic tasks** can be guaranteed as periodic tasks with $C_i = WCET_i$ and $T_i = MIT_i$
  
  \[
  \begin{align*}
  WCET &= \text{Worst-Case Execution Time} \\
  MIT &= \text{Minimum Interarrival Time}
  \end{align*}
  \]
SOFT aperiodic tasks

- Aperiodic tasks with SOFT deadlines should be executed as soon as possible, but without jeopardizing HARD tasks.

- We may be interested in
  - minimizing the average response time
  - performing an on-line guarantee
Periodic Scheduling (EDF)

\[ \tau_1 \quad C_1 = 1 \]

\[ \tau_2 \quad C_2 = 3 \]
Immediate service

\[ \tau_1 \]
\[ C_1 = 1 \]

\[ \tau_2 \]
\[ C_2 = 3 \]

**Response Time = 3**

**Deadline miss**
Background service

\[ \tau_1, C_1 = 1 \]
\[ \tau_2, C_2 = 3 \]

Response Time = 10
Exercise

3 min

• How can we improve response time?

- be creative!

![Graph showing response times with τ₁ and τ₂]
Aperiodic Servers

- A server is a kernel activity aimed at controlling the execution of aperiodic tasks.
- Normally, a server is a periodic task having two parameters:
  \[
  \begin{cases}
  C_s & \text{capacity (or budget)} \\
  T_s & \text{server period}
  \end{cases}
  \]

To preserve periodic tasks, no more than $C_s$ units must be executed every period $T_s$
Aperiodic service queue

- The server is scheduled as any periodic task.
- Priority ties are broken in favor of the server.
- Aperiodic tasks can be selected using an arbitrary queueing discipline.
Fixed-priority Servers

- Polling Server
- Deferrable Server
- Sporadic Server
- Slack Stealer
Dynamic-priority Servers

- Dynamic Polling Server
- Dynamic Deferrable Server
- Dynamic Sporadic Server
- Total Bandwidth Server
- Constant Bandwidth Server
Polling Server (PS)

- At the beginning of each period, the budget is recharged at its maximum value.

- Budget is consumed during job execution.

- When the server becomes active and there are no pending jobs, $C_s$ is discharged to zero.

- When the server becomes active and there are pending jobs, they are served until $C_s > 0$. 
RM + Polling Server

\[ \tau_1 \]

- \( C_1 = 2 \)

\[ \tau_2 \]

- \( C_2 = 1 \)

\[ \text{ape} \]

\[ \text{PS} \]

- \( C_s = 1 \)
- \( T_s = 5 \)

**Response Time = 8**
**PS properties**

- In the worst-case, the PS behaves as a periodic task with utilization $U_s = C_s/T_s$.

- **Schedulability**

  $$U_p + U_s \leq 1$$

- **RM**

  $$U_p + U_s \leq U_{lub}(n + 1) = (n + 1)(2^{1/n+1} - 1)$$
**PS properties**

- In the worst-case, the PS behaves as a periodic task with utilization $U_s = C_s/T_s$.

- Aperiodic tasks execute at the highest priority if $T_s = \min(T_1, \ldots, T_n)$.

- Liu & Layland analysis gives that:

$$U^{RM+PS}_{lub}(n) = U_s + n \left[ \left( \frac{2}{U_s + 1} \right)^{1/n} - 1 \right]$$
Worst case scenario

\[ T_s < T_i < 2T_s \]

\[ C_s = T_1 - T_s \]

\[ C_1 = T_2 - T_1 \]

\[ C_2 = T_3 - T_2 \]

\[ C_{n-1} = T_n - T_{n-1} \]

\[ C_n = T_s - C_s - \sum_{k=1}^{n-1} C_k \]
Computing $U_{ub}$ for $n$ tasks

\[ U_{ub} = U_s + \frac{T_2 - T_1}{T_1} + \cdots + \frac{T_n - T_{n-1}}{T_{n-1}} + \frac{2T_s - T_n}{T_n} \]

\[ U_{ub} = U_s + \frac{T_2}{T_1} + \cdots + \frac{T_n}{T_{n-1}} + \frac{2T_s}{T_n} - n \]

Defining $R_i = \frac{T_{i+1}}{T_i}$ and noting that $P = \prod_{i=1}^{n-1} R_i = \frac{T_n}{T_1}$

we can write:

\[ U_{ub} = U_s + \sum_{i=1}^{n-1} R_i + \frac{2(T_s / T_1)}{P} - n \]
Computing $U_{ub}$ for n tasks

$$U_{ub} = U_s + \sum_{i=1}^{n-1} R_i + \frac{2(T_s / T_1)}{P} - n$$

$$C_s = T_1 - T_s \quad \Rightarrow \quad \frac{C_s}{T_s} = \frac{T_1}{T_s} - 1 \quad \Rightarrow \quad T_s / T_1 = \frac{1}{U_s + 1}$$

$$K = \frac{2}{U_s + 1}$$

$$U_{ub} = U_s + \sum_{i=1}^{n-1} R_i + \frac{K}{P} - n$$
RM + PS schedulability

\[ U_{\text{lub}}^{RM+PS}(n \to \infty) = U_s + \ln \left( \frac{2}{U_s + 1} \right) \]

\[ U_{\text{lub}}^{RM+PS}(n) = U_s + n \left( K^{1/n} - 1 \right) \]
RM + PS schedulability

The Hyperbolic Bound revisited

\[ \prod_{i=1}^{n} (U_i + 1) \leq \frac{2}{U_s + 1} \]

- The periodic server is just one of the bunch!
RM + PS schedulability

Dimensioning the server parameters

- Let $P = \prod_{i=1}^{n} (U_i + 1)$

- Determine $U_{S_{max}}$ from $P \leq \frac{2}{U_{S}+1}$
  
  $U_{S_{max}} = \frac{2 - P}{P}$

- $T_S = \min(T_1, \ldots, T_n)$

- $C_S = U_{S_{max}} \times T_S$
RM + PS acceptance test

- Aperiodic job $J_a = (r_a, C_a, D_a)$
- Polling server $PS = (C_S, T_S)$
RM + PS acceptance test

- Aperiodic job $J_a = (r_a, C_a, D_a)$
- Polling server $PS = (C_S, T_S)$
- $R_a = C_a + \Delta_a + F_a(T_S - C_S) \leq D_a$
- $nxt = \left\lceil \frac{r_a}{T_S} \right\rceil T_S$
- $\Delta_a = nxt - r_a$
- $F_a = \left\lceil \frac{C_a}{C_S} \right\rceil - 1$

![Diagram showing RM + PS acceptance test with variables $r_a$, $nxt$, $f_a$, $d_a$, $R_a$, $\Delta_a$, $T_S - C_S$]
Deferrable Server (DS)

- Is similar to the PS, but the budget is not discharged if there are no pending requests.
- Keeping the budget improves responsiveness, but decreases the utilization bound.
RM + Deferrable Server

\[ \tau_1 \]
- \( C_1 = 2 \)

\[ \tau_2 \]
- \( C_2 = 1 \)

ape

DS
- \( C_s = 1 \)
- \( T_s = 5 \)

Response Time = 4
RM + Deferrable Server

utilization bound must be decreased to prevent deadline misses
Analysis of RM + DS

\[ U_{\text{lub}}^{RM + DS} (n) = U_s + n \left( \frac{U_s + 2}{2U_s + 1} \right)^{1/n} - 1 \]
RM + DS schedulability
RM + DS schedulability

The Hyperbolic Bound

\[ \prod_{i=1}^{n} (U_i + 1) \leq \frac{U_S + 2}{2U_S + 1} \]
RM + DS schedulability

Dimensioning the server parameters

• let $P = \prod_{i=1}^{n} (U_i + 1)$

• determine $U_s^{\text{max}}$ from $P \leq \frac{U_s + 2}{2U_s + 1}$

  $U_s^{\text{max}} = \frac{2 - P}{2P - 1}$

• $T_s = \min(T_1, \ldots, T_n)$

• $C_s = U_s^{\text{max}} T_s$
Exercise / challenge

Determine the schedulability under RM + DS of task set

<table>
<thead>
<tr>
<th></th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>t1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>t2</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>
RM + DS response time analysis

Critical instant

- Jitter analysis
- \( AJ_S = T_S - C_S \)

\[
R_i = C_i + \sum_{j<i} \left[ \frac{R_i + AJ_j}{T_j} \right] C_j
\]
RM + DS acceptance test

- Aperiodic job $J_a = (r_a, C_a, D_a)$
- Deferrable server $DS = (C_S, T_S)$
- $R_a = C_a + \Delta_a - C_0 + F_a(T_S - C_S) \leq D_a$
- $nxt = \left\lceil \frac{r_a}{T_S} \right\rceil T_S$
- $\Delta_a = nxt - r_a$
- $C_0 = \min(c_S(r_a), \Delta_a)$
- $F_a = \left\lceil \frac{(C_a - C_0)}{C_S} \right\rceil - 1$
RM + slack stealer

- periodic tasks can be delayed; their slack time can be pulled forward to serve an aperiodic task
- lowest response time of all fixed-priority servers
RM + slack stealer

Maintain slack time

• gaps need to be computed for all scheduling pts
  ➢ static: pre compute + lookup table
  ➢ dynamic: complex, but can handle (some) jitter
Is slack stealing optimal?

**NO!**

- **Theorem 5.3 (Tia-Liu-Shankar)** For any set of periodic tasks ordered on a given fixed-priority scheme and aperiodic requests ordered according to a given aperiodic queueing discipline, there does **not** exist any on-line valid algorithm that minimizes the average response time of the soft aperiodic requests.
Scheduling dilemma

What to do when an aperiodic job arrives at $t=2$?
- schedule immediately $\rightarrow$ you loose when job2 arrives
- wait $\rightarrow$ you loose when job2 does not arrive
Dynamic-priority Servers

- Dynamic Polling Server
- Dynamic Deferrable Server
- Dynamic Sporadic Server
- Total Bandwidth Server
- Constant Bandwidth Server
EDF + slack stealer

Slack available at the earliest possibility

- gaps need to be computed for all scheduling pts
  - dynamic: complex, but can handle (some) jitter
Total Bandwidth Server (TBS)

- It is a dynamic priority server, used along with EDF.

- Each aperiodic request is assigned a deadline so that the server demand does not exceed a given bandwidth $U_s$.

- Aperiodic jobs are inserted in the ready queue and scheduled together with the HARD tasks.
The TBS mechanism

- Deadlines ties are broken in favor of the server.
- Periodic tasks are guaranteed \( \text{if and only if} \)
  \[
  U_p + U_s \leq 1
  \]
Deadline assignment rule

- Deadline has to be assigned not to jeopardize periodic tasks.

- A safe relative deadline is equal to the minimum period that can be assigned to a new periodic task with utilization $U_s$:

\[ U_s = \frac{C_k}{T_k} \quad \Rightarrow \quad T_k = d_k - r_k = \frac{C_k}{U_s} \]

- Hence, the absolute deadline can be set as:

\[ d_k = r_k + \frac{C_k}{U_s} \]
Deadline assignment rule

- To keep track of the bandwidth assigned to previous jobs, $d_k$ must be computed as:

$$d_k = \max (r_k, d_{k-1}) + \frac{C_k}{U_s}$$
EDF + TBS schedule

\[
\begin{align*}
\tau_1 & : C_1 = 1 \\
\tau_2 & : C_2 = 3
\end{align*}
\]

\[\begin{array}{c}
0 \quad 2 \quad 4 \quad 6 \quad 8 \\
r_1 \quad r_2 \quad \text{ape} \quad d_1 \quad d_2
\end{array}\]

\[U_s = 1 - U_p = 1/4\]

\[
\begin{align*}
d_1 &= r_1 + C_1 / U_s = 1 + 2 \cdot 4 = 9 \\
d_2 &= \max(r_2, d_1) + C_2 / U_s = 9 + 1 \cdot 4 = 13
\end{align*}
\]

Giorgio Buttazzo – University of Pavia – 2002
Optimal TBS

Improve deadlines iteratively

do
    last = dᵢ;
    dᵢ = response_time(i);
while (dᵢ < last);

\[ C₁ = 1 \]
\[ C₂ = 3 \]

\( \tau_1 \)
\[ 0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \]
\( \tau_2 \)
\( 0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \)

\( \text{ape} \)
\( 0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \)

\( r₁ \)
\( r₂ \)
\( d₁ \)
\( d₂ \)

\[ d₁ = 9 \]
\[ d₁ = 7 \]
\[ d₁ = 6 \]
\[ d₁ = 3 \]
Exercise

8 min

• Schedule the jobs together with the periodic tasks using an Optimal TBS. Report the response times.

<table>
<thead>
<tr>
<th>Periodic tasks</th>
<th>C_i</th>
<th>T_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>t2</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jobs</th>
<th>a_i</th>
<th>C_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>j1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>j2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>j3</td>
<td>22</td>
<td>1</td>
</tr>
</tbody>
</table>
Problems with the TBS

- Without a budget management, there is no protection against execution overruns.
- If a job executes more than expected, hard tasks could miss their deadlines.
Types of overruns

- A task is said to be in overrun if the time demanded for execution exceeds the expected value according to which the task has been guaranteed.

- There are two types of overrun:

  **Execution overrun**
  
  A job executes more than expected

  **Activation overrun**
  
  A job arrives before than expected
Bandwidth enforcement

- It is a mechanism needed for degrading the QoS when a task demands more than the reserved bandwidth.

- If a task executes more than expected, its priority should be decreased (i.e., its deadline postponed).

- When a task experiences an overrun, only that task is delayed, so that the guarantee performed on the other tasks is preserved.
Constant Bandwidth Server (CBS)

- It assigns deadlines to tasks as the TBS, but keeps track of job executions through a budget mechanism.

- When the budget is exhausted it is immediately replenished, but the deadline is postponed to keep the demand constant.
CBS parameters

Given by the user

• Maximum budget: \( Q_s \)
• Server period: \( T_s \)

\[ U_s = Q_s / T_s \quad \text{(server bandwidth)} \]

Maintained by the server

• Current budget: \( c_s \) (initialized to 0)
• Server deadline: \( d_s \) (initialized to 0)
Basic CBS rules

• Arrival of job $J_k \Rightarrow$ assign $d_s$

  if $(r_k + c_s/U_s \leq d_s)$ then recycle $d_s$

  else \quad \begin{cases} 
  d_s = r_k + T_s \\
  c_s = Q_s 
  \end{cases}

• Budget exhausted \Rightarrow postpone $d_s$

  \begin{cases} 
  d_s = d_s + T_s \\
  c_s = Q_s 
  \end{cases}
Deadline assignment

\[ Q_s = 6 \]
\[ T_s = 12 \]
Budget exhausted

\[ Q_s = 3 \]
\[ T_s = 6 \]
EDF + CBS schedule

CBS: $Q_s = 2$, $T_s = 6$

Giorgio Buttazzo – University of Pavia – 2002
Exercise
6 min

- Schedule the jobs together with the periodic tasks using CBS with $C_S = 2$, $T_S = 8$. Report the response times.

<table>
<thead>
<tr>
<th>Periodic tasks</th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>t2</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jobs</th>
<th>$a_i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>j1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>j2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>j3</td>
<td>22</td>
<td>1</td>
</tr>
</tbody>
</table>
CBS properties

- **Bandwidth Isolation**

  If a task $\tau_i$ is served by a CBS with bandwidth $U_s$ then, in any interval $\Delta t$, $\tau_i$ will never demand more than $U_s \Delta t$.

- **Hard schedulability**

  A hard task $\tau_i$ $(C_i, T_i)$ is schedulable by a CBS with $Q_s = C_i$ and $T_s = T_i$, iff $\tau_i$ is schedulable by EDF.
CBS for all tasks

Look Ma: No WCET!

Basic CBS rules

- Arrival of job $J_k \Rightarrow assign d_s$
  
  $\text{if } (r_k + c_s / U_s \leq d_s) \text{ then recycle } d_s$

  $\text{else } \begin{cases} 
  d_s = r_k + T_s \\
  c_s = Q_s 
  \end{cases}$

- Budget exhausted $\Rightarrow$ postpone $d_s$
  
  $\begin{cases} 
  d_s = d_s + T_s \\
  c_s = Q_s 
  \end{cases}$

$U_{s1} + U_{s2} + U_{s3} \leq 1$