Project Proposal: Extending RTAI Linux with FPDS

Mike Holenderski

(version 0.2)

1 Introduction

The goal of this project is to extend the RTAI Linux with Fixed Priority Scheduling with Deferred Preemption (FPDS). The project consists of four exercises:

1. Get acquainted with RTAI Linux
2. Extend RTAI Linux with periodic servers
3. Extend RTAI Linux with deferable servers
4. Extend RTAI Linux with FPDS
5. Provide corresponding analysis

It involves implementing four experiments, starting with a simple ”Hello World” like task set example, and then adding on features until the RTAI kernel is extended with a full blown implementation of FPDS.

A laptop with pre-installed RTAI Linux is available for this project. [6, 4] are a good introduction RTAI.

1.1 Task model

For this project we assume the standard task model. A task $\tau_i$ is described by a phasing $\varphi_i$, (worst-case) computation time $C_i$ and deadline $D_i$, which means that a job generated by task $\tau_i$ at time $\varphi_i$ requires at least $C_i$ time of the resource before deadline $D_i$, i.e. before time $\varphi_i + D_i$.

A periodic task generates jobs at regular intervals and is further described by a period $T_i$, with job $\tau_{ij}$ representing the job released in the $j$th period at time $\varphi_i + jT_i$ with the deadline of $\varphi_i + jT_i + D_i$.

An aperiodic task generates a single job released at time $\varphi_i$ with the deadline of $\varphi_i + D_i$. When there is no confusion we will refer to a job generated by an aperiodic task $\tau_i$ simply as $\tau_i$.

In the remainder of this paper we assume that for all tasks $D_i = T_i$.

The $i$ index in $\tau_i$ refers to the priority of the task, where smaller $i$ represents higher priority. We assume there is a single task per priority and that there are altogether $n$ tasks in the task set. We further assume that the jobs do not self-suspend.

2 Get acquainted with RTAI Linux

Installing and calibrating RTAI Install RTAI Linux on a machine and calibrate it. This will require patching and building the linux kernel. Note that even though the laptop currently is running Linux patched with RTAI, this exercise will allow you to get aquatinted with the RTAI specific kernel options and building a kernel from source (if you are not familiar with it yet).
Example: simple task set   Once RTAI is successfully installed and calibrated, it is time to implement a simple example of a multimedia processing system. There are two main tasks in the system: a video task $\tau_v$ and a network task $\tau_n$. $\tau_v$ encodes raw frames and performs video content analysis on selected regions of the video frames, and $\tau_n$ wraps the encoded frames into packets and sends them over the network (see Figure 1).

We model these tasks with different parameters $C$ and $T$, and $\tau_n$ having a higher priority than $\tau_v$. In this exercise you will implement these two tasks with simple busy waiting loops:

```c
for i = 0 to N do
    skip
end
```

with $N$ chosen such that the loop will take approximately $C$ time. Tasks are independent (there is not inter task communication). Create a system which schedules these according to FPPS and FPNS.

3 Extending RTAI Linux with periodic servers

Servers were initially introduced to cope with aperiodic tasks [5, 3]. We would like to use servers to run periodic, sporadic and aperiodic tasks. A periodic server is specified by its period $T$ and capacity $C$. The server should replenish its capacity periodically and drop its remaining capacity if there is no pending workload. Each server can schedule its tasks using a different policy. Initially we implement FPPS.

Non-preemptive tasks require additional attention when the capacity of the server is smaller than the total workload of the pending tasks. To favor a simple implementation, before starting a non-preemptive task the server will check whether its capacity is sufficient to execute the task, by comparing the remaining capacity with the worst-case computation time of the task. If there is enough capacity, then the task will be scheduled, otherwise the remaining server capacity is reduced to 0 and the processor is released. Note that this requires the tasks to specify their worst-case computation time explicitly (currently RTAI does not include the computation time in its API).

We envision the following API, following the conventions of RTAI:

```c
typedef int Result;
Result rt_server_init(RT_SERVER* server, int capacity, ServerScheduler scheduler);
Result rt_server_task_make_periodic(
    RT_TASK* task,
    RT_SERVER* server,
    RTIME start_delay,
    RTIME period);
Result rt_server_task_make_oneshot(RT_TASK* task, RT_SERVER* server);
Result rt_server_make_periodic(RT_SERVER* server, RTIME start_delay, RTIME period);
```

$RT_SERVER$ is a structure, similar to $RT_TASK$ [4], storing the state of the server. It has to be allocated within the callers space and kept alive throughout the lifetime of the server.

$ServerScheduler$ is the scheduling policy for tasks within the server, and is defined as follows:

```c
typedef int ServerScheduler;
#define SERVER_SCHEDULER_FPPS 0
```
Initially we implement FPPS, and extend it with other scheduling policies in the following exercises.

Note that implementing servers may require modifying some of the RTAI code, e.g. the functions responsible for managing tasks.

**Example: tasks assigned to servers** Let \( \tau_n \) be a periodic task assigned to a periodic server \( \sigma_n \), with \( C(\tau_n) = C(\sigma_n) \) and \( T(\tau_n) = T(\sigma_n) \). Let \( \tau_v \) be a periodic task, and let \( \sigma_n \) and \( \tau_v \) be scheduled by the global scheduler according to FPPS, and let \( \tau_n \) be scheduled by \( \sigma_n \) scheduler according to FPPS.

### 4 Extending RTAI Linux with deferrable servers

We introduce servers with deferred replenishment. A server will keep its remaining capacity when there is no pending load, and replenish its capacity to \( C \) periodically with period \( T \). We envision the following API:

**Result** `rt_server_make_deferrable(RT_SERVER* server, RTIME start_delay, RTIME period);`

At this stage we would like to extend the server scheduling policies with FPNS:

`#define SERVER_SCHEDULER_FPNS 1`

**Example** Let \( \tau_n \) be a periodic task assigned to a deferrable server \( \sigma_n \), with \( C(\tau_n) = C(\sigma_n) \) and \( T(\tau_n) = T(\sigma_n) \). Let \( \tau_v \) be a periodic task with explicitly specified preemption points. Let \( \sigma_n \) and \( \tau_v \) be scheduled by the global scheduler according to FPDS, and let \( \tau_n \) be scheduled by \( \sigma_n \) scheduler according to FPNS.

### 5 Extend RTAI Linux with FPDS

We introduce FPDS scheduling [2, 1], both globally and within servers.

**Example** Let \( \tau_n \) be a periodic task assigned to a deferrable server \( \sigma_n \), with \( C(\tau_n) = C(\sigma_n) \) and \( T(\tau_n) = T(\sigma_n) \). Let \( \tau_v \) be a periodic task with explicitly specified preemption points. Let \( \sigma_n \) and \( \tau_v \) be scheduled by the global scheduler according to FPDS, and let \( \tau_n \) be scheduled by \( \sigma_n \) scheduler according to FPNS.

### 6 Extend servers to several tasks

Until now our examples assigned a single task to a server. To make the results of this project applicable in other applications, we would like the generalize the introduced mechanisms to servers holding several tasks. In this step we make sure that the implementation can support a system with several servers, each managing several tasks, with different scheduling policies on the global and local levels.

### 7 Incorporate communicating tasks

So far we assumed the tasks are independent. Our initial example in Section 2, however, tasks \( \tau_n \) and \( \tau_v \) are data dependent: \( \tau_v \) is activated upon raw frame arrival and \( \tau_n \) is activated when \( \tau_v \) finishes encoding the frames. We would like to extend our examples with communicating tasks, possibly exploiting the message passing and synchronization mechanisms offered by RTAI. To express the data-driven arrival of \( \tau_v \) we extend the periodic task specification with activation jitter and let \( \tau_v \) be a periodic task.

**Result** `rt_server_task_make_periodic(RT_TASK* task, RT_SERVER* server, RTIME start_delay, RTIME period, RTIME activationJitter);`
Let $\tau_n$ be an aperiodic task assigned to a deferrable server $\sigma_n$, with $C(\tau_n) = C(\sigma_n)$ and $T(\tau_n) = T(\sigma_n)$. Let $\tau_v$ be a periodic task, and let $\sigma_n$ and $\tau_v$ be scheduled by the global scheduler according to FPDS, and let $\tau_n$ be scheduled by $\sigma_n$ scheduler according to FPNS.

**8 Provide corresponding analysis**

We would like to evaluate the implementation and see how it fits with the existing analysis for servers [5, 3] and FPDS [2, 1].

**9 Optional: Exploit gain time**

In our implementation mentioned above, when a server does not have enough capacity to start a new task, it discards the remaining capacity. There are other methods for dealing with gain time, and all have an influence on the implementation and the analysis. We would like to discuss the alternatives and the objectives of our system, and implement those which we find well-suited.

**10 Contact**

Mike Holenderski, m.holenderski@tue.nl, 040 247 8364, HG 5.23

**References**


