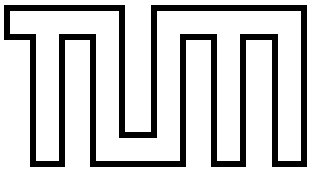


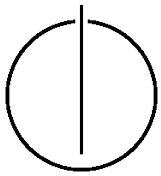
DEPARTMENT OF INFORMATICS

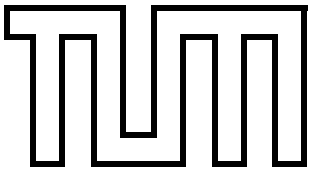
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of adaptive Mode-Switch-Algorithms in mixed critical Systems

Christoph Griesbeck





DEPARTMENT OF INFORMATICS

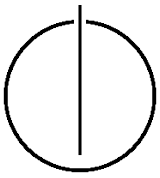
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of adaptive Mode-Switch-Algorithms in mixed critical Systems

Implementierung von adaptiven Mode-Switch-Algorithmen in gemischt kritischen Systemen

Author:	Christoph Griesbeck
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisors:	Biao Hu, M.Sc.
Submission date:	July 15, 2016



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

München, July 12, 2016 Christoph Griesbeck

Abstract

Modern systems nowadays often integrate functionalities of different safety-criticality on the same hardware. Therefore several specialized schedulers have been proposed for these so called mixed-criticality systems, to guarantee all deadlines of high critical jobs, while keeping the quality of service as high as possible.

A framework called SF3P was published to effectively simulate and analyze various scheduler set ups suitable for single core processors. However this framework didn't support mixed-criticality systems.

This paper covers the concepts and the implementation of several schedulers, which were designed for mixed-criticality systems and how SF3P has been updated to be able to simulate systems with tasks of multiple criticalities. Additionally the performance of these schedulers will be analyzed in respect to scheduling-overhead and quality of service.

Contents

List of Abbreviations

AMC	Adaptive Mixed Criticality scheme
C	Worst-Case Execution Time of a task
CPU	Central Processing Unit
D	Deadline of a task
DBF	Demand Bound Function
EDF	Earliest Deadline First queue
EDF-FFOB-A	EDF-VD scheduler with the standard FFOB scheme
EDF-FFOB-S	EDF-VD scheduler with the static FFOB scheme
EDF-VD	Earliest Deadline First-Virtual Deadlines scheduler
FFOB	on-the-Fly Fast Overrun Budgeting mode-switch scheme
FIFO	First-In First-Out queue
FP	Fixed Priority queue
HI	High
L	Criticality Level of a task
LO	Low
MCS	Mixed Cirticality System
OB	Overrun Budget, budget used in FFOB scheme
OP	Overrun Probability
PRNG	Pseudo Random Number Generator
QoS	Quality of Service
RAM	Random Access Memory
RTI-FP-F	FP scheduler with the standard FFOB scheme
RTI-FP-L	FP scheduler with the lightweight FFOB scheme
RTI-FP-S	FP scheduler with the static FFOB scheme
SBF	Supply Bound Function
SF3P	Scheduling Framework For Fast Prototyping
SMC	Static Mixed Criticality scheme
τ	Task set
T	Period of a task
TDMA	Time Division Multiple Access scheduler
VM	Virtual Machine
WBF	Workload Bound Function
WCET	Worst-Case Execution Time

1. Introduction

1.1. Motivation

Over the last century, it has been a major trend in the field of embedded systems to integrate components of different criticality onto one common platform. In this context, the criticality of a component is the level of assurance against failure[6]. These systems, which incorporate tasks of more than one criticality, are generally called mixed-criticality systems(MCSs)[6]. For example the current standard for software-safety in aviation(DO-178C), published by the *Radio Technical Commission for Aeronautics*(RTCA), specifies five levels of criticality from A to E[11]. While A refers to a task whose malfunction may lead to catastrophic behavior, E refers to a task whose malfunction has no impacts on the safety of the plane. For other industries similar standards have been published mostly using also five categories of criticality.

As certification authorities tend to be very conservative with estimating the properties of a system, especially the *worst-case execution times*(WCET) of tasks, manufacturers want these assumptions counting only for those high-critical tasks that shall be certified. Hence schedulers have been proposed with as many modes as criticality levels incorporated into the system, where in each mode only tasks of the same or higher criticality are executed. Additionally the allowed WCET of a task decreases with the current mode of the scheduler. Therefore, for jobs of lower criticality, more optimistic assumptions can be made, while not sacrificing the safety of the whole system. This is allowed, because as soon as the execution of all tasks isn't possible any more, all tasks of current criticality are dropped and all other tasks effectively run as if they used the hardware alone. Therefore higher WCETs, than originally estimated by the developer, are acceptable. However the quality of service, especially for jobs of low criticality, may suffer if the system spends to much time in higher mode. Nevertheless mixed-critical systems are highly effective, while maintaining the safety of the system. This has until now led to a considerable decrease of size, weight and power consumption of embedded systems.

To reduce the number of dropped jobs and the time spent in higher modes, while maintaining the schedulability of the whole system, is the goal of every scheduler. Furthermore the computing time needed by the scheduler itself shall be at a minimum. Because of these partly contrary requirements, tools for simulating, analyzing and comparing schedulers are needed. Although MCSs may consist of upto five different criticalities according to the standard, most work focuses on systems with only two, as every concept, developed for two levels of criticality, can easily be adapted for more levels. Thus most simulation tools also only support two levels of criticality.

1.2. Related Work

General The first basic work about verification of MCSs was done by Vestal in 2007[15]. Since then numerous papers have been published proposing different methods to analyse MCS, resulting in different approaches to schedule them. An complete overview over these is regularly updated by A. Burns and R. Davis and was lastly updated in January 2016[6].

EDF-VD To improve EDF scheduling of mixed-criticality systems, a new scheduler called *Earliest Deadline First Virtual Deadlines*(EDF-VD) was first proposed by Baruah et al.[4] and then further improved[1]. It relies mainly on predefined arbitrary shortening the deadlines of higher critical tasks to privilege them in LO-criticality modes and resuming them in HI-criticality mode. Therefore minimizing the risk of missed deadlines for higher critical jobs. By taking the task demand into account, shortening of the deadlines and therefore the schedulability can be fruther improved[7, 8].

AMC The first scheme, that used mode-switches with the realistic recurrent taskmodel in FP scheduled systems, was proposed by Baruah et al. and is generally referred to as *adaptive mixed-criticality scheme*(AMC)[2]. Two verions of this responsetime analysing approach were published in this paper called AMCrtb and AMCmax. This scheme was further improved by relaxing the strictness for scheduling by increasing task execution time thresholds[13]. Another publication based on the AMCrtb scheme recently proposed a so called bailout protocol, which uses the offline slack to efficiently switch the scheduler back to lower criticality modes[5].

FFOB A scheme called *on-the-fly fast overrun budgeting mode-switch scheme*(FFOB) has been presented by Hu et al. for systems scheduled by *Earliest Deadline First*(EDF)[11] and systems scheduled by *Fixed Priority*(FP)[12]. It is the first scheme that uses the online slack to improve the quality of service by delaying the switch to high modes, although this online computation results in increased overhead.

SF3P A framework called *Scheduling Framework For Fast Prototyping*(SF3P) was presented to the public by Gomez et al.[9] in 2014. This framework incorporated the safety and portability of a program run in user mode, while nevertheless being very efficient. However only basic schedulers like EDF or *Time Division Multiple Acces*(TDMA) were implemented. This framework was already two times extended to allow to simulate schedulers for multi-core processors and to test schedulers for MCSs[14, 16].

1.3. Problem Statement

Based on the original framework the goal of this thesis was to implement the new FFOB scheme for analyzing its performance. To be able to compare FFOB with schedulers it is based on, AMCmax was first implemented. After that, one version of EDF-VD was implemented on top of AMCmax. Lastly FFOB was implemented on top of both schedulers.

This thesis will first explain the theoretical principles of the different implemented schedulers in chapter 2.

Subsequently the implemented features will be elucidated in chapter 3. Foremost general enhancements to SF3P are explained in section 3.1. Afterwards further improvements, grouped by the schedulers, which are first requiring it, are examined in section 3.2, 3.3 and 3.4.

In chapter 4 the results of extensive simulations of the different schedulers in the extended version of SF3P are examined. They are split up into section 4.1, where the quality of service of the different schedulers will be analyzed, and section 4.2, where the performance of updating the overrun budget will be analyzed.

At last the achievements of this thesis will be summarized in chapter 5. Furthermore possibilities for future work will be outlined in this chapter.

In Appendix A several graphics showcase mechanisms of the FFOB scheme on the basis of a small example taskset. These might help understanding the concepts of FFOB and of the implementation.

In Appendix B several graphs present the exact results of the simulations, which were done for analyzing the quality of service of the different used schedulers.

2. Principles of implemented Schedulers

To understand what has been implemented, the following chapter will explain the principles of the different schedulers. In all following explanations a dual-criticality task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is being scheduled and the sporadic task model is used. Each task τ_i is a four-tuple consisting of its period, relative deadline, WCETs and criticality $(T_i, D_i, \vec{C_i}, L_i)$. Every task has two WCETs (C_i^L, C_i^H). While C_i^H is a fixed boundary to the execution time of HI-critical tasks, C_i^L is a more optimistic bound. By definition C_i^H is zero for tasks of LO-criticality. Furthermore the equation $C_i^L \leq C_i^H \leq D_i$ $(C_i^L \leq D_i)$ must always hold for obvious reasons. As short notation we denote $\tau^L = \{\tau_i \in \tau | L_i = LO\}$ as the subset of all tasks with $L \equiv LO$ and accordingly $\tau^H = \{\tau_i \in \tau | L_i = HI\}$.

The *demand bound function*(dbf) for a normal sporadic task τ_i is

$$dbf(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - D_i^L}{T_i} \right\rfloor \cdot C_i^L \tag{2.1}$$

and the corresponding *workload bound function*(wbf) for τ_i is:

$$wbf(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i}{T_i} \right\rfloor \cdot C_i^L \tag{2.2}$$

The *supply bound function*(sbf) for a dedicated uniprocessor with unit-speed, which is used in the following, and a given task set τ is:

$$sbf(\tau, \Delta) = \Delta \tag{2.3}$$

Furthermore the system is generally considered schedulable if the following two conditions hold:

$$\text{Condition LO: } \forall \Delta \geq 0 : \quad dbf_{LO}(\tau, \Delta) \leq sbf(\tau, \Delta) = \Delta \tag{2.4}$$

$$\text{Condition HI: } \forall \Delta \geq 0 : \quad dbf_{HI}(\tau^H, \Delta) \leq sbf(\tau^H, \Delta) = \Delta \tag{2.5}$$

2.1. Principles of AMC

The Adaptive Mixed Criticality scheme was introduced by Baruah et al. in 2011 and relies on fixed priority scheduling[2]. It is an advanced version of the previously published *static mixed-criticality scheme*(SMC), which is dominated by AMC in terms of schedulability. The authors additionally note, that scheduling a task set of more criticality-levels doesn't introduce any fundamental issues. The main idea of this scheme is monitoring

the execution times of all jobs and take action, if one exceeds its predefined worst case execution time.

In the AMC scheme the system initially is in LO-mode. As long as the system is in this mode, all jobs are scheduled by their tasks priority, as a standard FP scheduler would do. However while being executed, the execution time of each task is monitored. The special idea of this paper was to switch the system into HI mode if any job exceeds its respective C_i^L without signalling completion. Once the system is in HI-mode, only jobs of $L \equiv HI$ are executed. Of all remaining jobs still the one with highest priority is executed. Switching back to LO-mode was investigated in a previous paper about SMC[3] and therefore isn't investigated in the paper about AMC. However it is proposed to switch back whenever no job of HI-criticality is active. Furthermore this paper states that jobs of LO-criticality are only descheduled when a mode-switch to HI-mode occurs. However when the system switches back to LO-mode, this results in no guaranties that it is schedulable. Henceforth all LO-critical jobs are dropped when a mode-switch occurs. This also implies that jobs of LO-criticality, arriving while the system is in HI-mode, are instantly dropped.

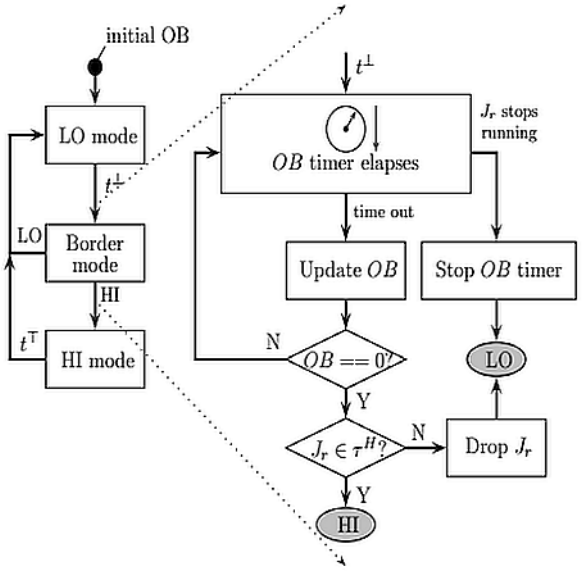
After presenting the AMC scheme, two methods called AMCrtb and AMCmax are proposed to determine if a given task set is schedulable under the AMC scheme. While AMCrtb (for response time bound) only considers the response time of tasks to determine the schedulability of the system, AMCmax uses stricter bounds by incorporating that HI-criticality tasks not always execute for their respective C_i^H.

2.2. Principles of EDF-VD

The Earliest Deadline First-Virtual Deadlines scheduling algorithm was first proposed by Baruah et al. in 2011[4] and was further improved by the same team[1]. However it was only applicable to implicit deadline task systems until A. Easwaran improved it[7]. As its name suggests it is based on the standard EDF scheduler and additionally the concept of AMC scheme. The main idea of this scheduler is to shorten the deadlines of HI-criticality tasks while the system is in LO-mode.

The dual-criticality system is initially in LO-mode(called level 1 in the original paper) and switches into HI-mode(called level 2), whenever a job exceeds its WCET without signaling completion. To improve scheduling, the authors came up with the idea of shortening the deadlines of HI-critical tasks while the system is in LO-mode. However when the system is switched to HI-mode, because a job has overrun, the normal deadlines are restored, thus creating a buffer for HI-critical jobs to not miss their deadlines. Because the system is scheduled by EDF, shortening the deadlines additionally privileges tasks of HI-criticality. How much the deadlines are shortened is computed by considering the overall utilization of LO-critical tasks and the one of HI-critical tasks in the original two papers.

Figure 2.1.: Overview of FFOB in the MCS[11]



Because, as before mentioned, the scheduler was only applicable to implicit deadline task systems, a new method to compute the virtual deadlines was proposed by P. Ekberg and W. Yi by expanding the concepts of dbfs[8]. Furthermore they use the famous Greedy algorithm for computing the exact virtual deadlines, which is illustrated in listing 2.1. This approach was further improved by A. Easwaran under the name *Earliest Carry-over Deadline First*(ECDF). His published algorithm, demonstrated in listing 2.2, iterates over the whole task set several times, decrementing or incrementing the virtual deadlines in each round where possible by one.

2.3. Principles of FFOB

The on-the-fly fast overrun budgeting mode-switch scheme is currently developed by Hu et al. and will be published this year[11]. It is based on the AMC scheme, respectively the EDF-VD approach, and therefore applicable with only minor changes to FP scheduled and EDF scheduled systems. The general idea is to determine the slack of the system and thus delaying the mode-switch.

The maximum slack is computed before the system is started and is called *overrun budget*(OB). It is reset to its initial value whenever the system is idle. When a task takes longer to finish than his WCET would allow him, it is allowed to be executed for the time of the OB. During this time the system is in a so called border mode and further decisions are made according to figure 2.1. If the task signals completion before the OB is fully used, the

Listing 2.1: Greedy algorithm for tuning low-criticality relative deadlines[8]

```
candidates ← {i|τ_i ∈ HI(τ)}
mod ←⊥
l_max ← upper bound for l in Conditions A and B
loop
  final ← true
  for l = 0, 1, ..., l_max do
    if ¬A(l) then
      if mod =⊥ then
        return FAILURE
      end if
      D_mod(LO) ← D_mod(LO) + 1
      candidates ← candidates \ {mod}
      mod ←⊥
      final ← false
      break
    else if ¬B(l) then
      if candidates = ∅ then
        return FAILURE
      end if
      mod ← argmax_{i∈candidates} (dbf_HI(τ_i, l) − dbf_HI(τ_i, l − 1))
      D_mod(LO) ← D_mod(LO) − 1
      if D_mod(LO) = C_mod(LO) then
        candidates ← candidates \ {mod}
      end if
      final ← false
      break
    end if
  end for
  if final then
    return SUCCESS
  end if
end loop
```

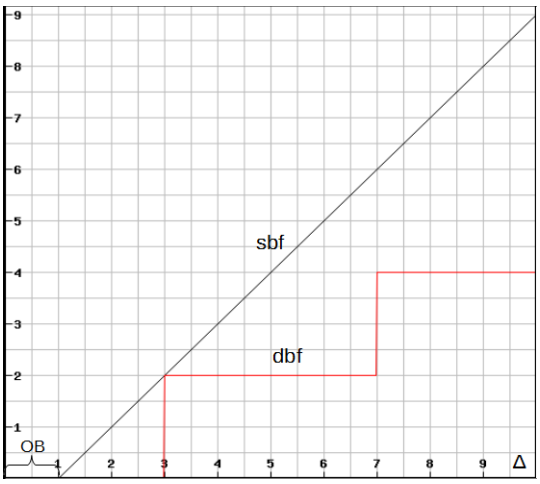
Listing 2.2: Earliest Carry-over Deadline First algorithm[7]

```
i ←⊥ and candidates ← {τ_i | τ_i ∈ H_τ}.
while True do
  feasible ← true.
  for t = 0...t_MAX do
    if Proposition 1 fails then
      If i =⊥, return failure.
      D_i^L = D_i^L + 1 and remove τ_i in candidates.
      i ←⊥ and break.
    end if
  end for

  for t_2 = 0...t_MAX and t_1 = 0...t_2 − min_{τ_i∈H_τ} {D_i − D_i^L} − 1 do
    if Theorem 2 fails then
      If t_1 = 0 or candidates = Φ, return failure.
      i = FINDCANDIDATE(candidates, t_1, t_2).
      D_i^L = D_i^L − 1.
      If D_i^L − 1 < C_i^L, remove τ_i in candidates.
      feasible ← false and break.
    end if
  end for
  If feasible is true, return success.
end while


function FINDCANDIDATE(candidates, t_1, t_2)
  Let DEM denote the excess demand at time instant t_2 (LHS of Theorem 2
      - t_2).
  result ←⊥, DIFF = 0, DEC = ∞.
  for Each task τ_i in candidates do
    if τ_i in case 2 and C_i^H − C_i^L ≥ DEM then
      if MOD(t_2 − t_1, T_i) − (D_i − D_i^L) < DEC then
        DEC ← MOD(t_2 − t_1, T_i) − (D_i − D_i^L).
        result ⊥ i and DIFF ← C_i^H − C_i^L.
      else if MOD(t_2 − t_1, T_i) − (D_i − D_i^L) = DEC and C_i^H − C_i^L > DIFF then
        result ← i and DIFF ← C_i^H − C_i^L.
      end if
    end if
  end for
  Return result.
end function
```

Figure 2.2.: Computation of the system's overrun budget



OB is decreased by the time used. If it hasn't signaled completion when the OB elapses, a new OB is computed. If this updated OB is equal to zero, it is determined if the task's criticality is LO or HI. If it is LO, the task only gets dropped and the system switches to LO-mode. If the task's criticality is HI, the system is switched to HI-mode and hence all jobs of criticality LO are dropped.

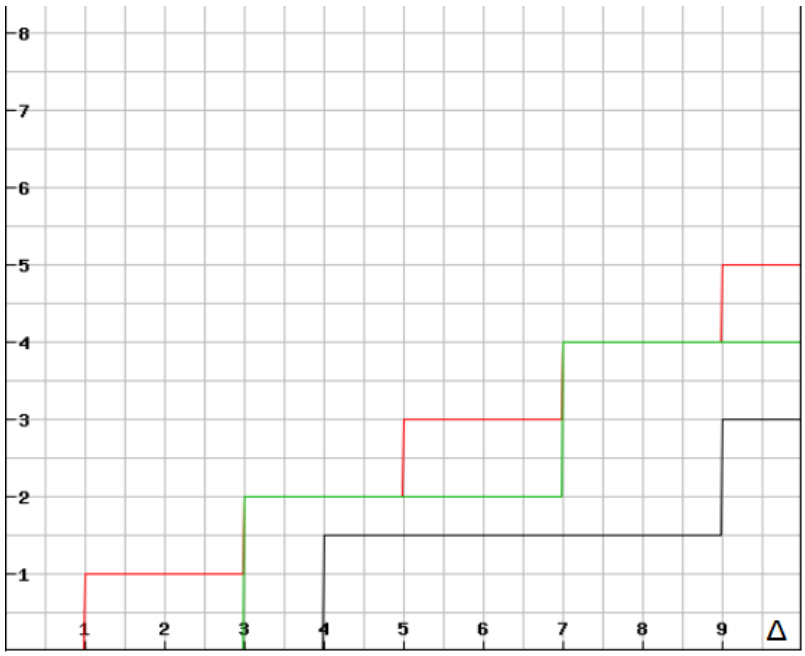
Because computing a new OB online, after it has elapsed to be zero, is computational expensive, it is also proposed to not do this. This approach is called static approach, as the OB is only initialized at startup and isn't updated if it elapses to be zero. However it is reset to its initial value, if an idle tick occurs and reduced when a task exceeds its WCET.

2.3.1. Principles of EDF-FFOB

The FFOB scheme is, as above mentioned, applicable to the EDF scheduler. Like the EDF-VD scheduler, all HI-critical tasks have a precomputed virtual deadline, which is used as long as the system is in LO-mode. The initial OB is computed on startup by comparing the sum of all tasks' dbfs, which effectively is the system's dbf, and the system's sbf. The system slack is the value by which the system can be delayed, while equations 2.4 and 2.5 hold. Practically it is the value by which the sbf, which is effectively a line through origin with gradient one, may be shifted to the right while the dbf is always smaller than it. An example with one task, whose dbf is drawn in red, and the system's sbf shifted to the right by 1, is demonstrated in figure 2.2.

If EDF-FFOB-S is used, which is the before mentioned static version, the updating of the overrun budget when it elapses to be zero is skipped. This results as shown in section 4.1 in lower QoS and lower scheduling overhead.

Figure 2.3.: The dbfs of two non-active tasks illustrated in black and green and additions if active in red



However if EDF-FFOB-A is used, the OB has to be updated when it elapses to be zero. In order to do so, the same principle, as in the initialization, is applied. However the dbf of active tasks is more complex as shown in equation 2.6,

$$dbf(\tau_i, \Delta) = \max(dbf(\tau_i, \Delta), Dmd^{bk}(\tau_i, \Delta, t)) \tag{2.6}$$

where $Dmd^{bk}(\tau_i, \Delta, t)$ is that

$$\begin{aligned} Dmd^{bk}(\tau_i, \Delta, t) = \max &\left(\left\lfloor \frac{\Delta}{r_i(t) + D_i^L - t} \right\rfloor, 1 \right) \cdot \min(C_i^L - e_i(t), 0) \\ &+ \min \left(\left\lfloor \frac{\Delta + \min(T_i, t - r_i(t)) - D_i^L}{T_i} \right\rfloor, 0 \right) \cdot C_i^L \end{aligned} \tag{2.7}$$

where $r_i(t)$ and $e_i(t)$ are the release time and the actual execution time of the latest released job of τ_i at t.

How the dbf may look like can be seen in figure 2.3. In black the dbf of a non-active task with $C_i^L = 1.5$, $T_i = 5$ and $D_i^L = 4$ is illustrated. In green is drawn how the dbf of another non-active task($C_i^L = 2, T_i = 4, D_i^L = 3$) could look like and in red the possible additions of Dmd^{bk}, if it were active.

Listing 2.3: Find minimal ρ^*[12]

```
ρ_l = 0, ρ_r = min(D_i − C_i^x), ∀τ_i ∈ τ
while ρ_r − ρ_l > ϵ do
  ρ* = (ρ_r + ρ_l)/2
  if Eqs.2.4 fails then
    ρ_r = ρ*
  else
    ρ_l = ρ*
  end if
end while
```

2.3.2. Principles of RTI-FP

The FFOB scheme is also applicable to FP scheduled MCSs. However it is a bit more complicated. Generally the OB is computed by the algorithm demonstrated in listing 2.3, where the resulting ρ^* is used as OB. The algorithm uses classical binary search to find the maximum overrun budget for which equation 2.4 holds, which therefore is the maximum OB usable with the given deadlines. However it is not possible to just sum up all tasks' dbfs to get the system's dbf and compare it with the system's sbf. Hence all tasks are sorted by their priority and then are each individually regarded. For every task its sbf is computed using equation 2.8, which is named *forward analysis*.

$$\begin{aligned} sbf(\tau_{i+1}^n, \Delta) = RT(sbf(\tau_0^n, \Delta), wbf(\tau_0^i, \Delta)), \quad \forall\, i \in \{1..n-1\} \\ wbf(\tau_0^i, \Delta) = \sum_{j=0}^{i} wbf(\tau_j, \Delta) \end{aligned} \tag{2.8}$$

where

$$RT(\beta, \alpha) = \sup_{0 \le \lambda \le \Delta} \{\beta(\lambda) - \alpha(\lambda)\} \tag{2.9}$$

To check if the system is scheduable, still equation 2.4 is decisive, but now has to be checked individually for every task.

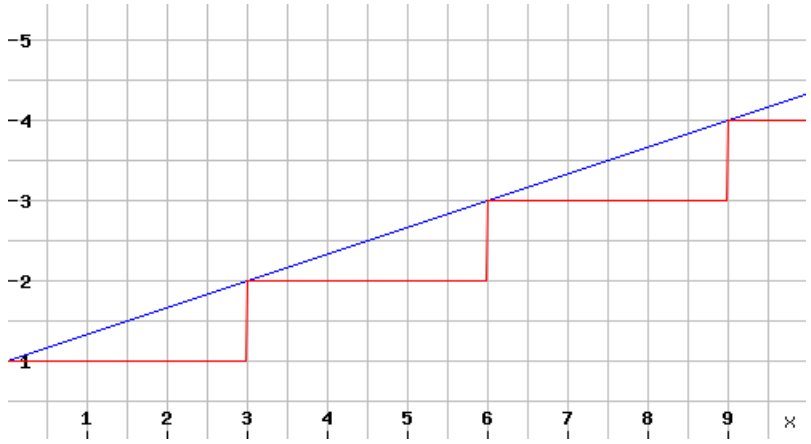
Like the dbf, the wbf is more complex for tasks that are currently active, but is similar as seen in equation 2.10,

$$wbf(\tau_i, \Delta) = \max(wbf(\tau_i, \Delta), Wrd^{bk}(\tau_i, \Delta, t)) \tag{2.10}$$

where $Wrd^{bk}(\tau_i, \Delta, t)$ is that

$$\begin{aligned} Wrd^{bk}(\tau_i, \Delta, t) = \max\left(\left\lfloor \frac{\Delta}{r_i(t) + T_i - t} \right\rfloor,\ 1\right) \cdot \min(C_i^L - e_i(t), 0) \\ + \min\left(\left\lfloor \frac{\Delta + \min(T_i, t - r_i(t)) - T_i}{T_i} \right\rfloor,\ 0\right) \cdot C_i^L \end{aligned} \tag{2.11}$$

Figure 2.4.: Approximation of wbf with a straight line



However this computation may be simplified by using straight lines to approximate the wbfs as illustrated in Figure 2.4. This results in all sbfs being straight lines too. Therefore comparing the sbfs with the corresponding dbfs gets by far easier, as only the gradients and y-intercepts are needed for computing the sbfs. This approach is called lightweight approach(RTI-FP-L), as it reduces the scheduling overhead by reducing the schedulability and the QoS.

3. Implementation details

3.1. General Additions and Improvements to SF3P

The scheduling framework for fast prototyping is a well designed framework, which was used as a base for implementing the in chapter 2 mentioned schedulers. A manual was published by Andres Gomez, which well explains the usage of the framework[10]. Unfortunately there wasn't any documentation for the source code available. It is written in C++, highly object orientated and well commented. However some improvements not directly related to the schedulers were made.

Memory related While tracking down a segmentation fault in the added code, a minor bug, which could have led to a segmentation fault, was found in the implemented operators for a struct named *timespec*. Timespec consists of two long long ints, one holding the amount of nanoseconds and one holding the amount of seconds. Every time the +, - or * operator was called, a new struct, which was allocated inside the function, was returned. Additionally to removing this bug, further operators like != were implemented. Another mistake that was corrected, was that many with new allocated variables were freed with free() instead of delete(), which is depreciated. Furthermore no destructors were implemented for any class, therefore leading to a minor memory leak, while being executed and have hence been added. However there was also some redundant code in the main scheduling functions, which was removed in the different already implemented schedulers. A problem, which was harder to track down, regarded the joining of threads. Although initially no errors happened, threads were multiply joined, which later led to memory corruption, because of duplicate freeing of before allocated variables. By only calling join from a central loop, this was also taken care of.

Randomization of Executiontime A major improvement to SF3P was made regarding its tasks. Initially only busy waiting and decoding a video were implemented as possible tasks. However it is fixed what video is decoded and busy-wait-tasks always executed for exactly their WCETs. Only because of inaccuracies in the time measurement, the execution times differed slightly by about $100\mu s$ per $10ms$. Therefore the execution time of busy-wait-tasks was randomized by using a *mersenne twister* as pseudo random number generator(PRNG). First it is used for deciding if the task shall overrun and afterwards for determining its concrete execution time. If the task shall overrun, its execution time is at minimum its C_i^L and at maximum two times its C_i^L. If it shall not overrun, its execu-

tion time lies between 0.6 times its C_i^L and its C_i^L. For seeding the mersenne twister, the current system time in microseconds is used.

Parser In addition the parser was heavily extended. Not only it now supports the added schedulers, but also multiple WCETs for tasks, which is essential for MCSs. Furthermore storing of the period of tasks was corrected, as initially the designated variable wasn't set. Additionally the parser isn't case sensitive at the detection of the scheduler anymore. Moreover the possibility to directly specify the OB in the input file was added to the parser.

3.2. Implementation of AMCmax

The most basic and therefore first implemented scheduler was AMCmax. Because this was the first implemented scheduler for MCSs for SF3P, most features were implemented while implementing it.

3.2.1. Runtime measurement for tasks

The first feature implemented was measuring the execution time of each job. Therefore a new class called *overrunChecker* was added. As every task has its own wrapper class called *Worker*, which does all management work around the task itself, this class is also responsible for calling the methods in the class *overrunChecker*. For each task an own overrun checker exists, whose id is 1000 times the id of the corresponding task and which measures the execution time of all jobs of this task.

Every time a job starts its execution, its worker fires the overrun checker. This results in entering the function *measureTask*, shown in listing 3.1, which measures the execution time of the job by using clock_nanosleep. The function first waits until the task is active and after that checks if joining was signaled. If not, nanosleep is invoked, which normally sleeps until the tasks WCET has elapsed. However it is possible to interrupt this sleeping by sending a signal to this thread, which is done if for example the task gets suspended. When sleeping is finished or interrupted the remaining time is stored in *remain*, and in *ret* it is signaled if sleeping was interrupted or not. Next the function checks if it shall reset, because for example the task has finished. If this isn't the case, the return value of nanosleep, stored in *ret*, is checked, if the sleeping was finished. If so the while-loop is exited and after resetting some values, an overrun gets signaled to the worker, which relays the signal to the scheduler.

The class *overrunChecker* can also be used for triggering the overrun event, when a deadline is missed instead of when a task overruns its WCET. Furthermore it supports returning the left over execution time for the current job.

Listing 3.1: Measurement of the execution time of one job

```
void OverrunChecker::measureTask(){
  int ret = -1;

  do{
    // wait for activation
    sem_wait(&sem_run);

    if (sem_trywait(&sem_join) == 0){
      return;
    }

    // wait for overrun or interrupt, remaining time in timeout
    sem_wait(&sem_timeout);
      ret = clock_nanosleep(HSF_CLOCK, 0, &timeout, &timeout);
      remain = timeout;
    sem_post(&sem_timeout);

    if(sem_trywait(&sem_reset) == 0){
      return;
    }
  }while(ret != 0);

  fired = false;

  // signal finished
  sem_post(&sem_end);

  //handle occurred overrun
  handleOverrun();
}
```

3.2.2. Dropping of Jobs

Another major feature that was integrated into the framework to support MCSs, is the dropping of jobs. Whenever a mode-switch occurs all LO-critical jobs have to be dropped. Therefore every worker, responsible for a task of LO-criticality, has to be signaled to drop its job. This is done by an extra function inside the scheduler-class, which loops over all active jobs. However the workers won't get the activate priority and therefore calculating time, before the next job of it starts its execution. Therefore the worker is left in a state where the previous job has only been suspend and canceling has only been signaled. This gets especially problematic, if the job was lastly suspended while the worker was doing some management and before the task actually got started. As it is unpredictable when the task and therefore the worker gets suspended and later canceled, it requires several work to ensure that everything is kept in a valid state. Especially the before mentioned overrun checker has to be managed right, so that no false overruns are reported. Additionally there had to be an interface for the task itself implemented, which allows canceling the task.

3.2.3. Implementation of AMCmax itself

Furthermore the AMC scheme itself still had to be implemented. The general scheduling for all event-based schedulers(e.g. FP, EDF, FIFO) was happening in a central method called schedule, which took action whenever an event, for example the arrival or the finish of a job, had happened. This function was the same for all event-based schedulers, as the order of execution was determined by a queue, which was sorted by e.g. the jobs deadlines if it was an EDF scheduler or by the priority if it was a FP scheduler. This design was kept and the first major addition was a second queue of the same type, which only held all HI-critical tasks.

Moreover as mentioned in subsection 3.2.1, a handler had to be implemented, which notices the signaled overrun and registers it as an event, which then can be handled by the main schedule function. This event then is handled by calling the function, which drops all LO-critical jobs and setting the queue containing only HI-critical jobs as the active queue, which represents the switch of the system into HI-mode. Furthermore the event is registered. Naturally, when a new job arrives and the system is in HI-mode, the jobs criticality-level is checked and the job is dropped if $L \equiv LO$.

Additionally whenever a job finishes, it is checked if it was the last active job and if the system is in HI-mode. If both is the case, the system is switch to LO-mode by setting the queue, containing all jobs, as active and registering this event too.

3.3. EDF-VD

Another implemented scheduler is the in section 2.2 explicated EDF-VD scheduler. Because it introduces the context of virtual deadlines, the queues, holding all active jobs

sorted by their deadlines, had to be modified. Thus the EDF-queue is now able to be operated in two modes. However the used mode has to be set at initialization and can't be switched at runtime. If it is operated in LO-mode, it considers the virtual relative deadlines instead of the normal relative deadline, which are considered in HI-mode. Therefore enabling the jobs being scheduled in one queue by their virtual relative deadlines, while the system is in LO-mode, and by their normal relative deadlines by the other queue, while the system is in HI-mode.

Furthermore to create the possibility to calculate the virtual deadlines for all jobs, an initialization function was added, which is called by the parser after parsing all tasks. This function gets a list of all tasks that will be scheduled, which is stored in a variable named *jobList*.

3.4. Implementation of FFOB

The last and most complex implemented scheme consisting of multiple schedulers is the FFOB scheme. It is implemented as an abstract class, which then is implemented by the several schedulers, based on EDF or FP, using it. The new feature introduced by it is the so called overrun budget. For computing it, the sbf of the system and the dbf and wbf of each task have to be computed, as explained in section 2.3. A Key decision for implementing the FFOB scheme was how to store each function. It was decided to store all functions as a vector of pairs of timespecs. Each pair denotes the coordinates of either one step of the dbf respectively wbf, or one delay in the sbf. The first field always is the x-coordinate of the step/delay, while the second field represents its size. Because the dbf/wbf/sbf might consist of up to three non periodic steps for each task, as demonstrated in figure 2.3, its size is three times the number of tasks.

3.4.1. Implementation of FFOB scheme itself

Because of the introduced overrun budget, whenever a new task arrives, its timeout is set to the sum of its WCET and the current OB. That is why every-time the OB changes, the timeouts of all tasks have to be updated. Anytime a task exceeds its WCET, but finishes before the OB is exhausted, the OB has to be decreased by the amount used. Furthermore whenever an idle tick occurs the OB has to be reset. Hence the maximum OB is stored in a second variable on initialization.

When the OB elapses to be zero decisions according to figure 2.1 of section 2.3, have to be made. This first includes updating the OB, which is explained in detail in the following sections. After that it is checked if the OB is smaller than one microsecond; not zero because of inaccuracies in the algorithm of listing 2.3. If it is true and the criticality of the overran job is HI, all LO-critical jobs are dropped and the system mode is switched to HI by setting the queue containing only HI-critical jobs as active. Else only the currently active job is dropped and the system stays in LO-mode.

3.4.2. Computation of DBF and WBF

Key for computing the overrun budget is computing the dbf and wbf for all tasks. Therefore two extra methods were implemented. The one for computing the dbf is illustrated in listing 3.2, while the one for computing the wbf is similar as the wbf itself is very similar to the dbf.

Obviously, for computing the dbf for all tasks, the function loops over all tasks, which were stored on initialization in *jobList*, as described in section 3.3. After that C_i^L and D_i^L are stored for further usage. Then if the currently processed job is an active job, both summands of Dmd^{bk}, from equation 2.6 and illustrated in red in figure 2.3, are computed and stored, else those two fields are left empty. However not the exact values are used for the second field, but the relative step size. This later makes computing the y value of the dbf at this point and therefore comparing the dbf with the system's sbf easier. It shouldn't stay unnoticed, that it must always be checked, that no underflow happens, as unsigned data types are used. Afterwards the aspect of the offline dbf is also stored. After the dbf has been computed for all tasks, all steps whose height is zero are erased by setting their x-coordinate(first field) to zero. This later helps, especially when computing the sbf out of the wbf.

3.4.3. Implementation with EDF

Because the FFOB scheme has been implemented as an abstract class, only initializing and updating the OB has to be handled in the subclasses. In case of EDF-FFOB-S, this updating of the OB is rather simple, as the OB just has to be set to zero.

However initializing the OB is in both versions the same. First the previously mentioned *jobList* is sorted by the relative virtual deadlines of the tasks. After that the dbf for each task can easily be computed using a similar approach to that seen in listing 3.2, although without the part for currently active tasks and without checking for entries that are zero. Subsequently the interval is chosen in which the sbf and the dbf shall be compared. To be sure that all cases are considered, the length of the regarded interval is two times the size of the highest relative virtual deadline. Then the final dbf inside this interval is computed, as shown in listing 3.3 by adding all occurrences of each step of the dbf and afterwards sorting the whole dbf by the first element of each entry. Finally it is iterated over each dbf-entry as shown in listing 3.4, thereby accumulating the second values and computing the difference of this sum and the corresponding first entry. The minimum of all these differences is the maximum OB of this MCS.

For updating the OB first the vector for holding the dbf is created and filled with zeros. Afterwards, the in section 3.4.2 explained function for computing the dbf is used. Then again the interval length is determined and the dbf extended, similar to listing 3.3. After deleting all entries with zero and sorting the whole dbf by the first field of each pair in ascending order, the current OB can be computed as shown before in listing 3.4.

Listing 3.2: Computation of the dbf for all jobs

```
void FFOB::computeDBF(vector<pair<timespec, timespec>> &dbf){
  uint i = 0;
  timespec c, d ,t;
  for(Worker* w : jobList){
    c = w->getWCET(0);
    d = w->getCriteria()->getRelativeVirtualDeadline();

    if(activeQueue->deleteRunnable(w->getId())){
      activeQueue->insertRunnable(w);//reinsert the runnable

      t = w->getCriteria()->getPeriod();

      //first summand
      timespec tmp = TimeUtil::getTime() - w->getReleaseTime();
      dbf[i].first = d>tmp ? d - tmp : TimeUtil::Millis(0);
      dbf[i].second = w->getRemainingTime() > overrunBudget ? w->
          getRemainingTime() - overrunBudget : TimeUtil::Millis(0);//min(
          remain , 0)
      i++;

      //second summand
      dbf[i].first = (t>tmp ? t - tmp : TimeUtil::Millis(0)) + d;
      dbf[i].second = dbf[i-1].second;
      i++;
    }
    else{
      i+=2;
    }

    dbf[i].first = d;
    dbf[i].second = c - dbf[i-2].second;
    i++;
  }

  for(i = 0; i < dbf.size(); i++){
    if(dbf[i].second == TimeUtil::Millis(0))
      dbf[i].first = TimeUtil::Millis(0);
  }
}
```

Listing 3.3: Extension of dbf

```
timespec max = (*max_element(dbf.begin(), dbf.end())).first;
max = 2*max;

for(uint i = 0; i < jobList.size()-1; i++){
  t = jobList[i]->getCriteria()->getPeriod();
  vector<pair<timespec,timespec>> tmp(TimeUtil::convert_ms(max - dbf[i
      ].first)/TimeUtil::convert_ms(t));

  for(uint k = 1; k*t+dbf[i].first <= max; k++){
    tmp[k-1].first = k*t + dbf[i].first;
    tmp[k-1].second = dbf[i].second;
  }

  dbf.insert(dbf.end(), std::make_move_iterator(tmp.begin()), std::
      make_move_iterator(tmp.end()));
}
```

Listing 3.4: Comparison of dbf and sbf in EDF scheduled systems

```
timespec c = TimeUtil::Millis(0), d, ob = TimeUtil::Millis(1000);

for(uint i = 0; i < dbf.size(); i++){
  c = c + dbf[i].second;
  d = dbf[i].first;

  ob = d-c < ob ? d-c : ob;
}
```

3.4.4. Implementation with FP

Additionally to the subclasses using the FFOB scheme with EDF schedulers, there are subclasses for using the FFOB scheme with FP schedulers. When the schedulers are initialized, the *jobList*, which holds all tasks, is sorted by the priority of the tasks, which is a prerequisite for computing the sbf. Because of the lack of time, only updating the OB, when it has elapsed to be zero, has been implemented and not initializing it. For the RTI-FP-S scheduler, updating is rather simple, as the OB only has to be set to zero.

However for both, RTI-FP-F and RTI-FP-L, first the in section 3.4.2 explained functions for computing the dbf and wbf are used and afterwards the algorithm demonstrated in listing 2.3 is used for computing the OB. However checking if equation 2.4 holds and the therefore needed preparation of the wbf, is done differently.

RTI-FP-F In case of RTI-FP-F, the wbf is prepared, as demonstrated in listing 3.3 for the dbf, as again an interval of double the length of the highest virtual deadline is considered. For checking if equation 2.4 holds, it is generally looped over all tasks. For every task all steps in its dbf are checked, as this is sufficient to guarantee that the sbf is smaller than the dbf. Therefore first the sbf of the current task has to be computed. For accomplishing this, the entries of the previous task's wbf are appended to an array holding the sbf of the previous task. However the first value of all entries has to be at minimum the value of the OB, as not until then the system starts computing. Sorting the whole obtained array results in the sbf of the current task. However, because no two jobs can be executed in parallel, all colliding sbf entries have to be delayed until the other job has finished.

Afterwards it has to be checked, that the dbf is always smaller than the sbf. This is done by the principle shown in listing 3.5, which is similar to listing 3.4. However, all entries of the sbf until the current examined position have to be added onto the temporary value too, as they are effectively the delays of the sbf. This is done inside the while-loop of listing 3.5, where *ptr* points to the actual sbf element and *sbfPtr* points to the last sbf element plus one. Because only the delay until the current position has to be included, it is checked inside the loop if the current position is in the interval of a delay. If this is true, instead of adding the full delay, only the delay that happened until the current position is added. At last it is checked if the obtained value is greater than the current position minus the OB, as this would mean that the dbf would be greater than the sbf at this point. This has to be done for all three parts of the dbf. Because the second and third entry have to be considered several times, as they are periodic, they have to be checked inside a loop, which loops over all occurrences inside the before specified interval.

RTI-FP-L As RTI-FP-L relies mainly on approximating the wbf and therefore the sbf, first the wbf has to be approximated by straight lines as shown in figure 2.4. The approximated wbf is stored as a vector of pairs of double and unsigned int. Each pair denotes one straight line, where the double represents the grade and the y-intercept is stored as an unsigned int, because the y-intercept of the wbf has to be greater or equal zero. The

Listing 3.5: Comparison of dbf and sbf in FP scheduled systems

```
c = c + dbf[i].second;

while(sbf[ptr].first < dbf[i].first && ptr < sbfPtr){
  if(sbf[ptr].first + sbf[ptr].second > dbf[i].first){
    c = c + (dbf[i].first - sbf[ptr].first);
  }else{
    c = c + sbf[ptr].second;
  }

  ptr++;
}

if(dbf[i].first - _overrunBudget < c)
  return false;
```

slope is computed by dividing the size of the step through the period of the task. If the wbf of a non active task is computed, the y-intercept is equal to the WCET of the task. Else it has to be checked if the y-intercept is dictated by the steps added by Wrd^{bk} from equation 2.10 or not. If not it is set to zero, else it is computed by using the previously computed gradient and the higher point of the first step of the wbf.

To check if an overrun budget is valid and consequently equation 2.4 holds, equation 2.4 once again has to be checked individually for every task. First the sbf for the current task has to be computed. This is done by simply subtracting the slope of the wbf of the previous task from its sbf and by adding the corresponding y-intercepts. Afterwards it is checked that all three dbf entries are lower than the sbf and that the slope of the dbf is smaller than the slope of the sbf. As the sbf is approximated as straight lines, its sufficient to check the steps of the dbf once.

4. Evaluation

All simulations were done inside a Virtual Machine(VM) with Debian Jessie as operating system. The host is a Laptop with an Intel i7-3630QM(up to 3.4GHz) as CPU and 8GB RAM. 4 out of 8 logical kernels are available to the VM, although, as later mentioned, the simulation is limited to one. Furthermore 3GB are available to the VM. For all time measurement it shall be noted that everything is computed with an accuracy of 1 nanosecond, although the call to clock_gettime is accurate to about $250ns$. However all included values for the quality of service are rounded to milliseconds and for performance measurements to microseconds for better readability.

4.1. Quality of Service

4.1.1. Setup

EDF For analyzing the performance of the FFOB scheme on top of an EDF scheduler, a total of 40 tasksets were simulated with EDF-FFOB-A, EDF-FFOB-S and EDF-VD with an overrun probability(OP) of 0.1, 0.01 and 0.001. Each taskset was simulated for 15 minutes for each setting. Furthermore each randomly generated taskset consisted of 8 tasks, half of them LO-critical, with a period between 20 and 1000 milliseconds and WCETs ranging from 1 to 500 milliseconds. Measured for the QoS was the number of dropped jobs, the number of mode-switches and the time the system had spent in HI-mode.

FP Furthermore the FFOB scheme on top of a FP scheduler was also simulated for being analyzed. A similar setup was used, however the tasksets consisted of 20 tasks and the number of LO-critical tasks wasn't strict 10. Simulated were RTI-FP-F, RTI-FP-L, RTI-FP-S and AMCmax.

4.1.2. Problems

When accumulating the numbers, simulating all EDF-schedulers takes 90 hours and all FP-schedulers 120 hours, which in total is 210 hours or nearly 9 days. Therefore at first always two simulations were done in parallel. However while lastly doing the simulations with an OP of 0.001, it was found out, that running them in parallel heavily increased the number of overrunning jobs and consequently falsified the results. This is due to inaccuracies in the execution time of Busy-Wait tasks, which heavily increases with the lack of processing power and nevertheless a relatively exact overrun checker. Though because

Table 4.1.: Results of the simulations of EDF

OP = 0.1	EDF-FFOB-A	EDF-FFOB-S	EDF-VD
Dropped Jobs	11,408	180,195	769,699
Mode-switches	3223	47,870	179,180
Time in HI-mode(ms)	144,840	1,746,703	9,048,702

OP = 0.01	EDF-FFOB-A	EDF-FFOB-S	EDF-VD
Dropped Jobs	2453	28,674	155,161
Mode-switches	697	15,561	58,989
Time in HI-mode(ms)	55,073	445,484	2,056,001

OP = 0.001	EDF-FFOB-A	EDF-FFOB-S	EDF-VD
Dropped Jobs	421	6616	32,193
Mode-switches	77	1783	9920
Time in HI-mode(ms)	3549	47,757	301,828

of the lack of time and the increased impact on simulations with lower OP, it was only possible to rerun the simulations with an OP of 0.001 and not all. Therefore the results of simulations with an OP of 0.1 and 0.01 are not fully reliable. Especially as the influence on the results increases with decreasing numbers and consequently falsifies the ratios.

4.1.3. Results

The exact results of the simulations are attached as several graphs in appendix B. More significant are the accumulated numbers for each scheduler and for each OP, which can be seen in table 4.1 for EDF schedulers and table 4.2 for FP schedulers. Each one line inside a table thereby summarize the information of one diagram in the appendix. These summarized values will be used in the following, because single results are varying to much, despite the fact that every simulation lasted 15 minutes.

Earliest Deadline First When looking at the plain accumulated numbers of table 4.1, it is indisputable that the FFOB scheme increases the quality of service for LO-critical tasks. Furthermore it is clear that the static approach is by far less effective than the advanced, but however is still considerably more effective than plain EDF-VD. To put it in numbers, the number of dropped jobs for the static approach is 14 times higher than for the advanced approach, while being about one fifth of plain EDF-VD. Nearly the same results can be gathered for the time spent in HI-mode, while the number are even slightly higher when looking at the number of mode-switches. The later might be explained,

as in the FFOB scheme an overrunning job of LO-criticality only gets dropped, while in contrast a mode-switch happens in EDF-VD.

When using relative numbers the improvements are even more obvious. The results mean, using for example the time spent in HI-mode with a system's OP of 0.1, that under EDV-VD on average one fourth of the simulation time was spent in HI-mode, while its only 0.4 percent that were spent in HI-mode on average under EDF-FFOB-A. Regarding the number of dropped jobs, approximately 3.1 million LO-critical jobs were scheduled. This means that under EDF-VD nearly one fourth of all LO-critical jobs were dropped, while under EDF-FFOB-S its only one twentieth and with EDF-FFOB-A its even 0.3 percent or about one three hundredth.

Another point to mention is, that the numbers compared between the different overrun probabilities are decreasing by about the factor 0.2, whereas the OP decreases by the factor 0.1. One minor cause could be, that the expected count of not overrunning jobs before one overruns is computed as the number of all jobs plus one divided by the number of overrunning jobs plus one. Furthermore it can be assumed that if less jobs overrun, the count of resets of the overrun budget, because an idle tick occurred, is larger and therefore the QoS increases.

Table 4.2.: Results of the simulations of FP

OP = 0.1	RTI-FP-F	RTI-FP-L	RTI-FP-S	AMC
Dropped Jobs	24,311	45,450	172,327	378,230
Mode-switches	5314	7948	33,167	86,082
Time in HI-mode(ms)	442,293	566,642	1,680,102	5,683,333

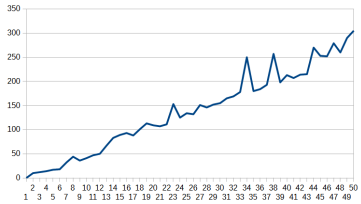
OP = 0.01	RTI-FP-F	RTI-FP-L	RTI-FP-S	AMC
Dropped Jobs	2780	5443	27,733	87,370
Mode-switches	624	1297	5535	24,998
Time in HI-mode(ms)	33,957	66,891	258,004	1,365,426

OP = 0.001	RTI-FP-F	RTI-FP-L	RTI-FP-S	AMC
Dropped Jobs	553	1640	6253	18,040
Mode-switches	158	379	1244	4688
Time in HI-mode(ms)	5589	15,954	55,451	246,303

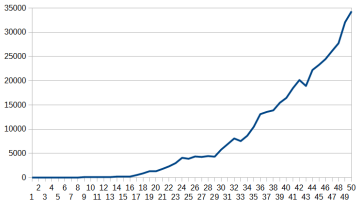
Fixed Priority When looking at the FP-scheduled tasksets, the FFOB scheme also has a positive effect on the QoS. However the numbers are distributed differently. The static approach is generally a bit more than 3 times better than plain AMCmax, while the light weight approach is again 4 times better than the static approach. However the exact approach is only 2 times better than the lightweight approach, especially regarding the num-

Figure 4.1.: Duration of updating the OB depending on the taskset's size

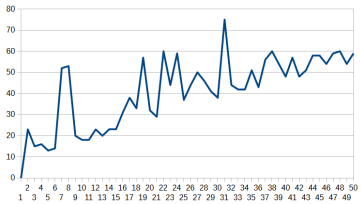
(a) EDF-FFOB-A



(b) RTI-FP-F



(c) RTI-FP-L



ber of mode-switches, where the improvements are even lower. On the contrary the static approach ones again especially decreases the number of mode-switches, for the same reason as mentioned before for edf-scheduled systems. This leads to the assumption, that the exact approach can very well be replaced by the lightweight approach.

Compared to EDF schedulers, where the exact approach is 14 times better than the static one, for FP schedulers its only 9 times better. Furthermore in EDF the exact approach is about 66 times better, while in FP its only 15 times better. Therefore it can be concluded, that although the improvements of the FFOB scheme are large for MCSs scheduled by fixed priority schedulers, they are even larger for systems scheduled by earliest deadline first schedulers.

The same effect, that the results don't scale with the same factor as the OP, can be observed for the FP schedulers, although the effect is a little bit smaller than for EDF schedulers. The presumed reasons for this phenomenon are the same as mentioned in the previous paragraph for EDF scheduled systems.

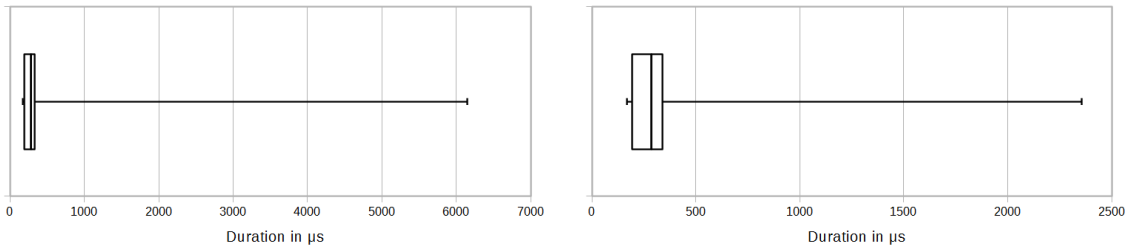
4.2. Performance of Schedulers

For analyzing the performance of the different schedulers when updating the overrun budget, initially a taskset consisting of 50 tasks was used. After each one minute during simulation, the taskset's size was decreased by one and a new simulation was started. For better results the overrun probability was set to 0.3333 and at minimum 200 updates were used for each taskset. For tasksets consisting of less than 5 tasks an OP of 1 was used to generate the needed updates.

4.2.1. Performance with EDF

When looking at the performance of EDF-FFOB-A, the amount of time needed for updating the OB increases nearly linear with the number of tasks being scheduled, as seen in Figure 4.2a. It starts by only 10 microseconds and increases to a maximum of 305 microseconds on average for 50 tasks. This is very reasonable, as computing the dbf as

Figure 4.3.: Duration of updating the OB in EDF-FFOB-A with 49/50 tasks



explained in section 3.4.2 and computing the OB, as described in section 3.4.3, are both linear in time complexity. Only extending the dbf is element of $\mathcal{O}(n*m)$, where m denotes the number of jobs that arrive in the considered interval. Because the interval presumably rises with the number of tasks being scheduled, this computation is not exactly linear in time complexity. Furthermore, as shown in the Whisker-Box-Plots of figure 4.3 for the tasksets consisting of 49 and 50 tasks, the variance in general is relatively small, although some rare extreme values happen.

4.2.2. Performance with FP

For FP scheduled MCSs, two schedulers were proposed, which update the overrun budget at runtime. Their results are demonstrated in figure 4.2b and 4.2c and obviously differ by a large extend. While both start by about 10 microseconds for updating the OB for one task, like EDF-FFOB-A also does, RTI-FP-F takes on average 34265μs for updating with 50 tasks, while RTI-FP-L only takes 60μs and therefore is by a factor of over 500 faster.

When looking at the graph of RTI-FP-F, the graph looks at least of quadratic time complexity. Because the same methods are used for computing the dbf and wbf as RTI-FP-L does and the same scheme for extending the dbf is used as in EDF-FFOB-A, but however the consumed time is by far larger, checking that equation 2.4 inside algorithm 2.3 holds is the dominating factor. This has also been verified by measuring the individual sections of the updating method. For being able to check if this equation holds in general, it has to be checked for every task individually. Computing the sbf of a single job is in $\mathcal{O}(m + nm \cdot log(nm))$, because first the extended wbf has to be appended, which consists of m elements depending on the size of the interval, and after that the whole sbf has to be sorted. The actual checking if equation 2.4 holds furthermore is of time complexity $\mathcal{O}(m)$ as again it is looped over the whole interval. Therefore the total complexity is $\mathcal{O}(n \cdot (nm \cdot log(nm) + 2m))$ or summarized as $\mathcal{O}(n^2 m \cdot log(nm)))$. This now clearly indicates a time complexity worse than $\mathcal{O}(n)$ and therefore must be considered not practical for larger tasksets. Particularly as 34 milliseconds is in now way acceptable for a scheduler to make its decision. Furthermore when looking at figure 4.5, updating the OB in

Figure 4.5.: Duration of updating the OB in RTI-FP-F with 49/50 tasks

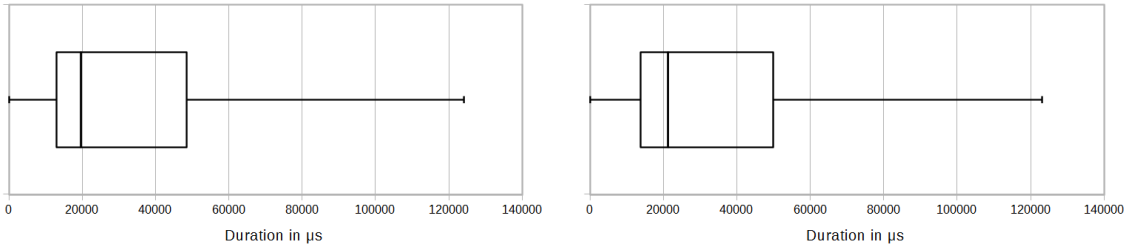
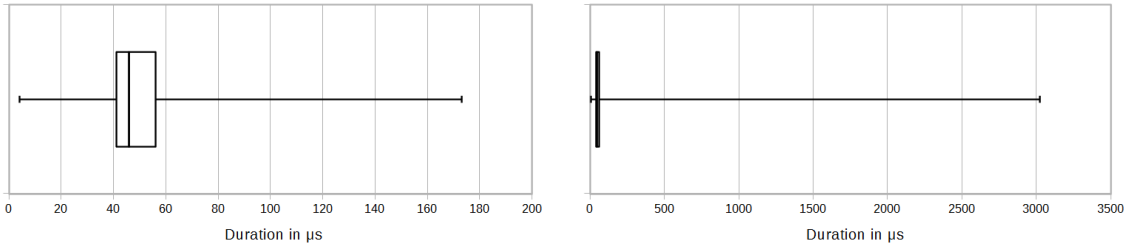


Figure 4.7.: Duration of updating the OB in RTI-FP-L with 49/50 tasks



RTI-FP-F underlies a high variance and ranges up to 120 milliseconds, which is also not desirable.

When looking at the graph of RTI-FP-L, the graph seems to indicate a time complexity smaller than linear, e.g. logarithmic. However computing the dbf and wbf is of linear time complexity, as previously mentioned. Moreover checking if a given overrun budget is valid and approximating the wbf, as described in section 3.4.4, are both of linear time complexity too. Therefore the whole updating process is also of linear time complexity. It shouldn't stay unnoticed that although this lightweight approach and EDF-FFOB-A are both of linear time complexity, RTI-FP-L is about 5 times faster on average than EDF-FFOB-A for large tasksets. Moreover the variance is also even lower compared to EDF-FFOB-A, as the Whisker-Box-Plots in figure 4.7 for RTI-FP-L and in figure 4.3 for EDF-FFOB-A indicate.

Furthermore it should be noted, that algorithm 2.3 is of logarithmic time complexity regarding the maximum possible overrun budget. Especially for RTI-FP-F, where checking if an overrun budget is possible is costly, this is also a factor determining the time needed for updating the OB.

5. Conclusions and Future Work

5.1. Future Work

Simulation As always there are numerous ideas, what could be done in the future. The most obvious one is to do more elaborate simulations, because the results used in this thesis are, as already mentioned in section 4.1.2, not 100% reliable. First of all an otherwise idle sever should be used, to minimize interference of other processes, for simulating the tasksets in the future, as the job's execution time is highly sensitive to interferences, while measuring its execution time isn't. Moreover more tasksets and an increased duration of the simulation would significantly reduce variations inside the results. Furthermore simulations figuring out the influence of the size of the maximum overrun budget and the influence of the largest period, which determines the considered interval, on the performance of updating the overrun budget could be done and be helpful for improving the scheduler by decreasing its overhead.

SF3P However SF3P can also be further improved. For example the pseudo random number generator used for generating the execution time of each job, as described in section 3.1, can be better seeded. Although it is highly unlikely that two jobs' execution time is computed at the same time, the system time isn't changing enough to be considered a good seed. This could be improved by using the thread id and/or the WCET to xor with the first seldom changing bits of the system time. Additionally more realistic types of tasks, like encoding/decoding a video, could be implemented.

Furthermore fixing the often occurring segmentation faults inside the simfig tool would highly improve the users experience. Particularly as already for a bit longer simulations(1sec) with more than 10 tasks, its nearly unusable. Adding markings for overran jobs would further improve the tool. Moreover the number of dropped jobs, the count of mode-switches and the time spent in HI-mode could be added to the central statistics instead of being stored inside the scheduler and just printed to the console.

Further minor improvements could address the parser currently used for parsing the input xml-file. One improvement could be to implement a possibility to specify the overrun probability of all tasks inside the input file, replacing the currently hardcoded one. Moreover the input data could be checked in more detail for valid values, leading to more meaningful errors instead of just segmentation faults.

FFOB Naturally the implementation of the FFOB scheme can be further improved too. The most obvious improvement would be, to also be able to calculate the overrun budget for FP scheduled MCSs. Furthermore, being able to compute the virtual-deadlines of HI-critical tasks would be a great enhancement. As the results have shown, updating the OB is relatively costly. By being able to directly compute the maximal OB for a job, the whole binary search of algorithm 2.3 would get obsolete and a significant performance boost, especially for RTI-FP-F, would be achieved. Moreover while implementing the approximation of the wbf for RTI-FP-L, it was found out that the wbf of an active task can be approximated by two different straight lines. The one of them that isn't used right now would increase the OB, yet endangering the schedulability of the whole system. However this could be used purposefully for increasing the OB and therefore the QoS, while fully maintaining the schedulability of the system. This is particularly interesting, as RTI-FP-F in its current state has a to big overhead for being practically useful.

5.2. Conclusion

The goal of this thesis, integrating state-of-the-art schedulers into the scheduling framework for fast prototyping, was achieved. In order to do so, new features were implemented, therefore reducing the work needed for integration of further schedulers in the future. Additionally this thesis presents the principles of these state-of-the-art schedulers, which are based on earliest deadline first and on fixed priority. Moreover the schedulers were evaluated separately in terms of performance and quality of service. For EDF schedulers, the new schedulers significantly improve the QoS for LO-critical jobs, as demonstrated in section 4.1.3, while introducing only a small overhead, as shown in section 4.2.1. Hence the new schedulers are a great improvement in the fields of embedded systems. Meanwhile for FP schedulers, also significant improvements are made by the new schedulers, which is shown in section 4.1.3. However the exact approach has proven not practically useful in section 4.2.2, because of a huge overhead, which additionally lead to large delays of jobs. Nevertheless the lightweight approach reduced this overhead to nearly zero, while achieving almost the same results for the QoS as the exact approach and by a large extend better results than the static approach.

Appendix

A. Demonstration of FFOB at an example

Table A.1.: An example taskset with two tasks used for the following figures

	C_i	D_i	T_i
τ_1	2	3	4
τ_2	1	4	5

Figure A.1.: The dbf, wbf and sbf of τ_1

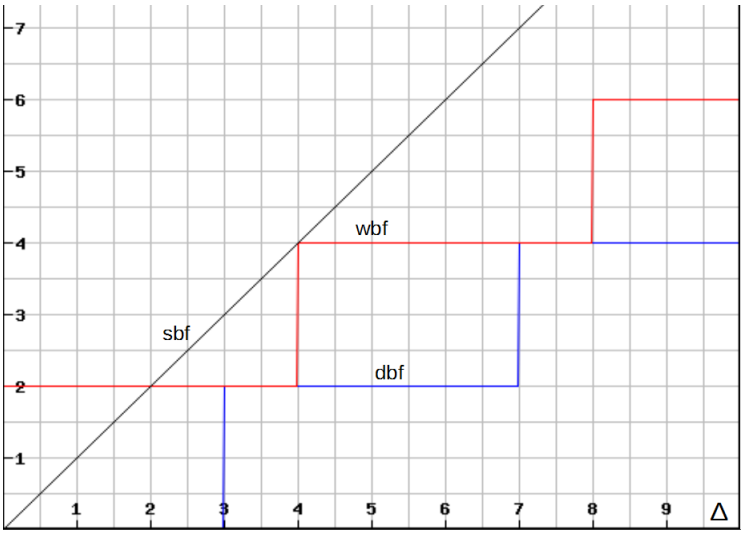


Figure A.2.: The dbf, wbf and after τ_1 leftover sbf of τ_2

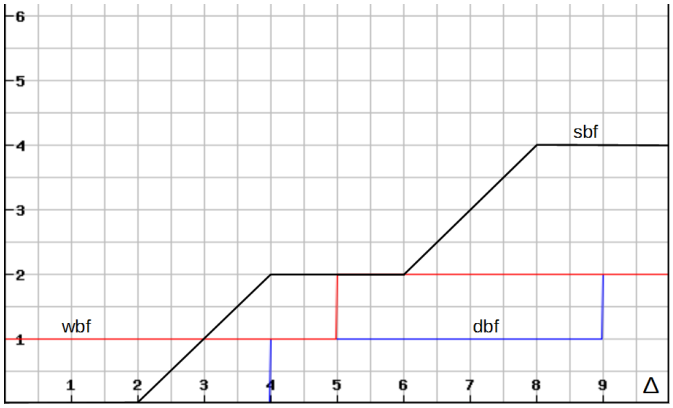
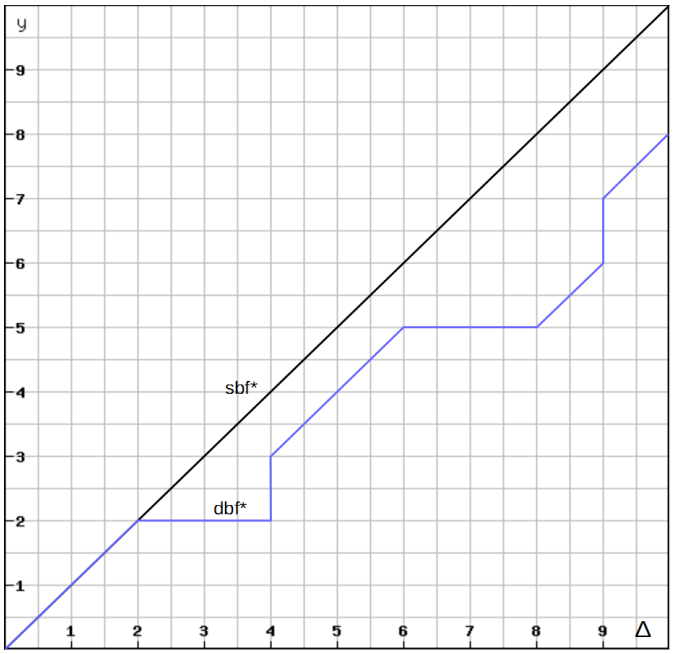
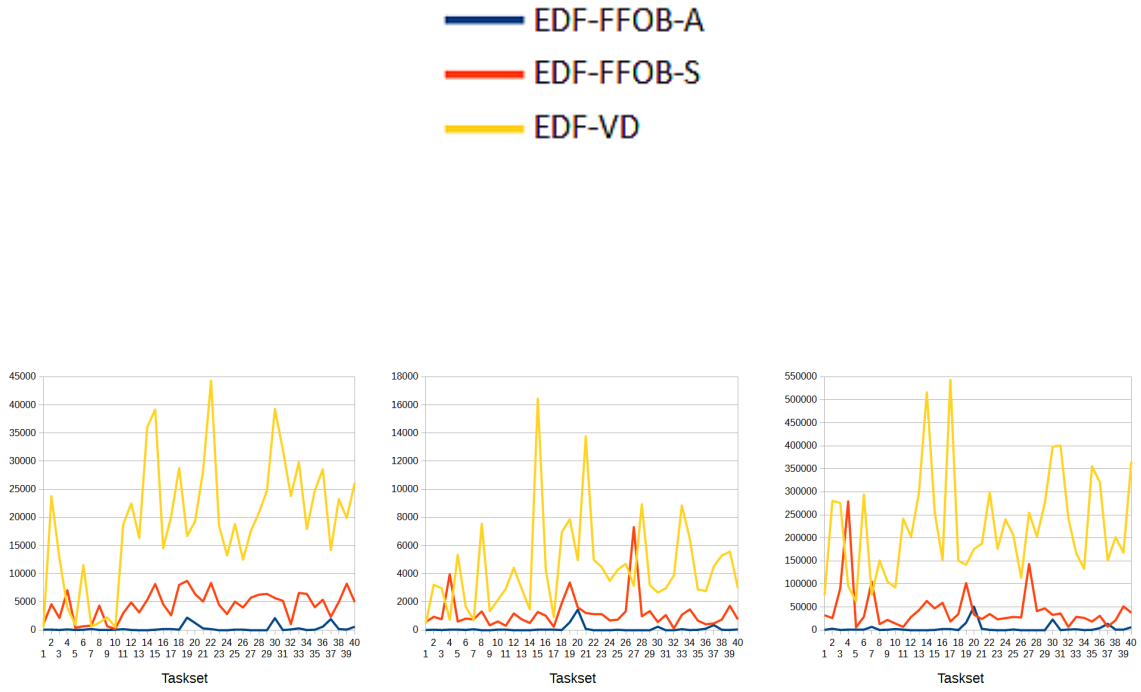


Figure A.3.: The sbf and dbf used by the program for τ_2



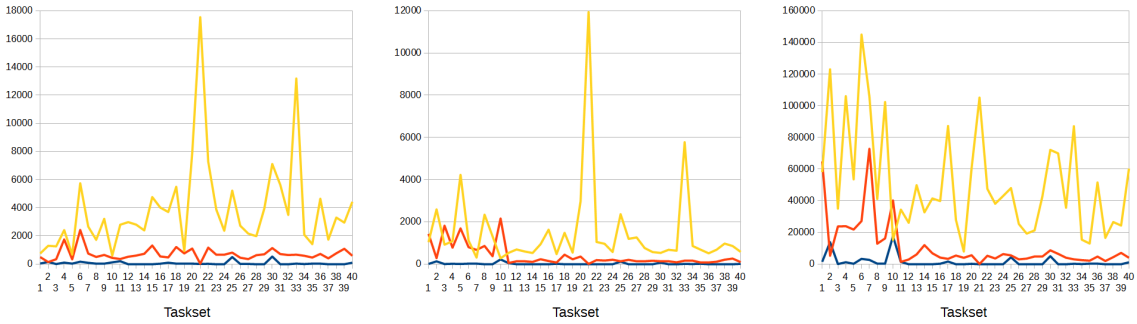
B. Results for the Quality of Service

B.1. Results of EDF-schedulers for each taskset grouped by OP



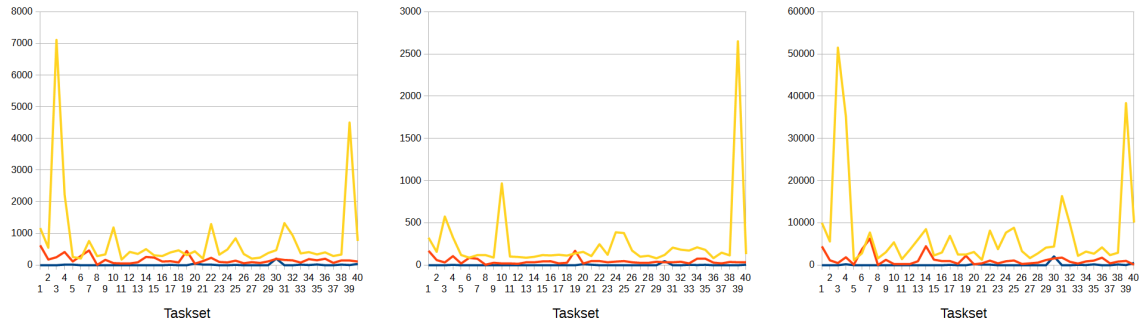
(a) Number of dropped jobs (b) Number of mode-switches (c) Time in HI-mode in ms

Figure B.1.: Results for an overrun probability of 0.1



(a) Number of dropped jobs (b) Number of mode-switches (c) Time in HI-mode in ms

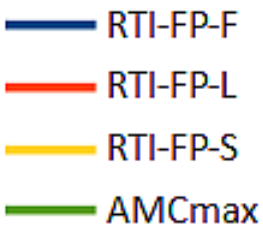
Figure B.2.: Results for an overrun probability of 0.01

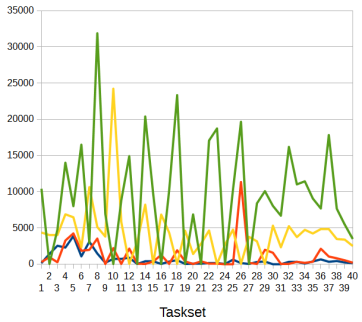


(a) Number of dropped jobs (b) Number of mode-switches (c) Time in HI-mode in ms

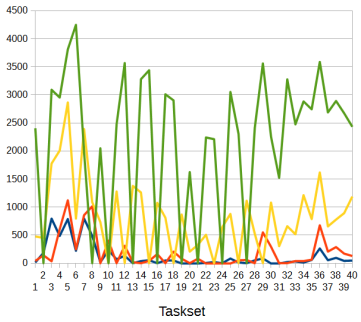
Figure B.3.: Results for an overrun probability of 0.001

B.2. Results of FP-schedulers for each taskset grouped by OP

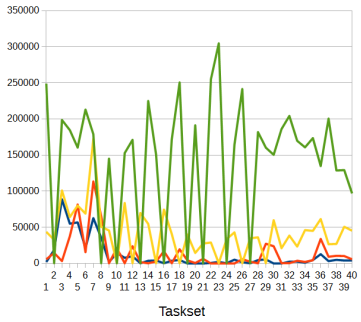




(a) Number of dropped jobs

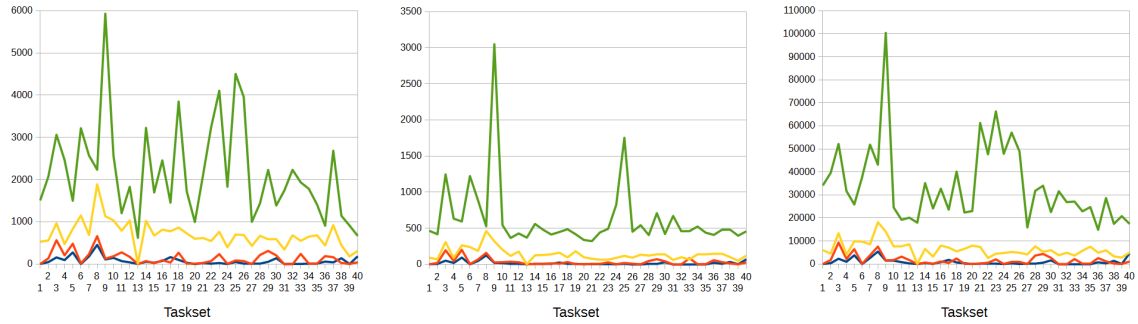


(b) Number of mode-switches



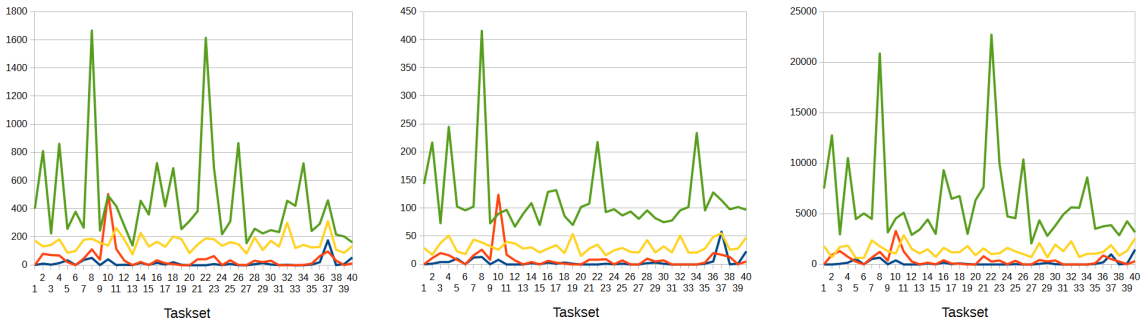
(c) Time in HI-mode in ms

Figure B.4.: Results for an overrun probability of 0.1



(a) Number of dropped jobs (b) Number of mode-switches (c) Time in HI-mode in ms

Figure B.5.: Results for an overrun probability of 0.01



(a) Number of dropped jobs (b) Number of mode-switches (c) Time in HI-mode in ms

Figure B.6.: Results for an overrun probability of 0.001

Bibliography

[1] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 145–154, July 2012.

[2] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43, Nov 2011.

[3] Sanjoy Baruah and Alan Burns. *Implementing Mixed Criticality Systems in Ada*, pages 174–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[4] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. *Mixed-Criticality Scheduling of Sporadic Task Systems*, pages 555–566. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] I. Bate, A. Burns, and R. I. Davis. A bailout protocol for mixed criticality systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 259–268, July 2015.

[6] Alan Burns and Rob Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013. Sixth edition, 1/8/2015.

[7] A. Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 78–87, Dec 2013.

[8] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 135–144, July 2012.

[9] A. Gomez, L. Schor, P. Kumar, and L. Thiele. SF3P: a framework to explore and prototype hierarchical compositions of real-time schedulers. In *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pages 2–8, Oct 2014.

[10] Andres Gomez. *SF3P: A Scheduling Framework For Fast Prototyping*.

[11] Biao Hu, Kai Huang, Pengcheng Huang, Lothar Thiele, and Alois Knoll. On-the-fly fast overrun budgeting for mixed-criticality systems. In *International Conference on Embedded Software (EMSOFT)*, October, to appear 2016.

[12] Biao Hu, Lothar Thiele, Pengcheng Huang, Kai Huang, and Alois Knoll. Online mode-switch procrastination in fp-scheduled mixed-criticality systems. Soon to be published, 2016.

[13] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 155–165, July 2012.

[14] Lukas Sigrist. Implementation and evaluation of mixed-criticality scheduling algorithms for multi-core systems. Semesterthesis, Swiss Federal Institute of Technology Zurich, jan 2014.

[15] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243, Dec 2007.

[16] Felix Wermelinger. Implementation and evaluation of mixed criticality scheduling approaches. Semesterthesis, Swiss Federal Institute of Technology Zurich, jun 2013.