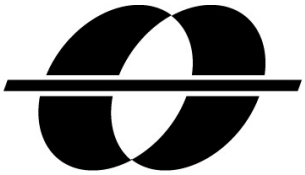

MÄLARDALEN UNIVERSITY
SWEDEN

Towards Adaptive Hierarchical Scheduling of Real-time Systems

Author:
Nima Moghaddami Khalilzad
nmi09001@student.mdh.se

Supervisor:
Moris Behnam
Examiner:
Thomas Nolte

School of Innovation, Design and Engineering (IDT)
Mälardalen University
Västerås, Sweden

April 11, 2011

Abstract

Hierarchical scheduling provides predictable timing and temporal isolation; two properties desirable in real-time embedded systems. In hierarchically scheduled systems, subsystems should receive a sufficient amount of CPU resources in order to be able to guarantee timing constraints of its internal parts (tasks). In static systems, an exact amount of CPU resource can be allocated to a subsystem. However, in dynamic systems, where execution times of tasks vary considerably during run-time, it is desirable to give a dynamic portion of the CPU given the current load situation. In this thesis we present a feedback control approach for adapting the amount of CPU resource that is allocated to subsystems during run-time such that each subsystem receives sufficient resources while keeping the number of deadline violations to a minimum. We also show some example simulations where the controller adapts the budget of a subsystems.

If we allocate CPU only based on subsystems demand and don't take into account the availability of the resource, timing guarantees of the lower priority subsystems (using a priority based scheduler in the global level) will be violated in the overload situations. In such a situation the high criticality modules should be superior to the low criticality modules in receiving resources. In this thesis, in the extension of our adaptive framework, we propose two techniques for controlling the CPU distribution among modules in an overload circumstance. First we introduce the notion of subsystem criticality and then distribute CPU portions based on the criticality level of subsystems.

Keywords: adaptive, hierarchical scheduling, feedback control, PI controller, overload control, Times tool

Acknowledgements

I would like to thank my supervisor Moris Behnam who helped me a lot during the thesis, especially in the control theory part of the thesis which he dedicated lots of time and effort. I am thankful for all interesting discussions in our meetings.

In addition, I would like to express my gratitude to Thomas Nolte who suggested me this interesting topic for my thesis, helped me throughout the thesis and supported me in publishing scientific papers from my thesis work. He provided me with several great papers and ideas. I am really grateful for all his feedbacks and comments on my work.

Moreover, I would like to acknowledge Mikael Åsberg who dedicated lots of his time guiding me especially in the simulation part of the thesis. He recommended me to use his model which he had developed in Times tool. It was a great solution for my simulation problem.

Finally, I am grateful to my beloved wife Arefeh for her love and endless support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Embedded real-time systems become increasingly more complex, making it difficult to combine hard real-time guarantees with efficient use of system resources. When run-time behavior of tasks in a complex real-time system is difficult to predict, the feedback scheduling concept can be used as a powerful tool for adapting scheduling to the task's requirements. For example a decoder task of an H264 stream can experience more than five times execution time variation depending on the video content [6]. Furthermore, the feedback scheduling is useful in systems that tasks are added or removed dynamically during run-time. Scheduling parameters can be adapted during run-time such that tasks get a better service in response to their request for the shared resources. Although a variety of techniques are available based on feedback scheduling, a suitable technique should be designed given the context of our Hierarchical Scheduling Framework (HSF) [30].

The HSF provides a modular way for scheduling and guarantying timing constraints of real-time tasks [15, 21]. The HSF can be illustrated using a tree structure in which each node is responsible for scheduling its children using resources received from its corresponding parent node. Each child provides the parent with parameters such as period and budget (the subsystem interface), and parents schedule their children according to the subsystem interface parameters. In doing so, we can achieve a component based abstraction which reduces complexity in designing a compositional real-time system. Resource efficient interface variables can be found, for example, by assuming a fixed period for subsystems and trying to find a minimum possible value for the budget in which the system is schedulable [35]. In this thesis, a feedback mechanism is introduced for online control of the interface parameters in a HSF. The goal of the presented approach is to adapt the budget of subsystems during run-time to achieve an efficient CPU utilization in comparison with systems having pre-assigned fixed interface parameters, especially when tasks within a subsystem experience a considerable change in their execution time. Given a particular subsystem period, the subsystem budget should be kept to a minimum while at the same time minimizing the number of potential deadline misses within a predetermined time-interval.

While, a number of studies have been conducted on overload scheduling [10, 8, 33, 14], scheduling of mixed criticality systems in overload situation are also investigated in [29]. Since none of these works have been applied to hierarchical scheduling, in this thesis we study some applicable techniques that can be applied in the context of our HSF [30].

The contributions of this thesis are the design of the feedback control system for dynamic adaptation of resource parameters in the HSF, simulation studies investigating the performance of our solution, and two methods for handling CPU overload in our Adaptive Hierarchical Scheduling Framework (AHSF).

1.2 Related Works

Related works of this thesis can be categorized in three groups: hierarchical scheduling, feedback scheduling and overload scheduling.

1.2.1 Hierarchical Scheduling

Since Deng and Liu [15] presented a two level hierarchical scheduling framework, there has been a growing attention for using hierarchical scheduling in complex real-time systems. Schedulability analysis for the two level frameworks is presented by Kuo and Li [20]. For EDF-based global schedulers analysis is presented by Lipari and Baruah [23, 22]. In addition, the virtual processor model is presented in [28, 35]. Then, based on this mode schedulability analyses under fixed priority scheduling [3, 24] and EDF [38, 35] are studied. While all of the aforementioned works allocate static CPU portions to the subsystems, we introduce an adaptive HSF which dynamically allocates CPU to the subsystems.

1.2.2 Feedback Scheduling

Feedback scheduling has been used in scheduling of control tasks for acquiring predictable performance when execution time of tasks are subjected to sudden changes [11]. Model Predictive Controllers (MPC) are scheduled using a feedback loop [18]. In [34] feedback-based scheduling is used in the real-time memory garbage collector. Feedback scheduling applied to reservation-based algorithms and a complete mathematical analysis is presented in [2]. In [13] a two level controller is proposed to share resources among a pipeline of tasks and satisfy Quality of Service (QoS) requirements. Scheduling of tasks was investigated in the stochastic domain and a two block controller was suggested [12]. In [1] a two-level feedback controller in the context of a reservation technique is introduced, where application level and system level QoS are improved based on bandwidth adaptation. In [37] optimizing techniques are used for controlling the CPU utilization in multiprocessor systems. Stankovic et al. have applied feedback control techniques in distributed systems [36] and they have proposed local and global level feedback controllers. Lu et al. introduced a Proportional Integral Derivative (PID) controller which controls CPU utilization requests based on miss ratio feedback [25]. They continued their work and presented a two-feedback loop system [26]. None of the aforementioned techniques have been applied in the context of HSF.

1.2.3 Overload Scheduling

de Niz et al. presented a scheme for protecting temporal isolation of high criticality tasks in mixed criticality systems [29]. In their scheme a low criticality task cannot interfere with a high criticality tasks. In [27] each task, in addition to a criticality value, has a mandatory and an

optional part. In overloaded situations a set of task parts are chosen that maximizes the overall value of the system. In [9] authors by introducing an elastic task model, showed that how tasks can adapted themselves to different quality of services. Their proposed approach suggests that in overloaded situations instead of rejecting a new task by reducing the utilization of other tasks, system lets the new task to use the CPU. In [32] a technique for dealing with overload situation using (m, k)-firm guarantee is proposed. The approached is suggested for real-time control tasks that can tolerate occasional deadline misses. In the (m, k)-constrained model, m out of k consecutive jobs should meet their deadlines. For example a (1, 1)-constrained system is a hard real-time system, however, a (3, 4)-constrained system is a soft real-time system in which system can tolerate one deadline miss every four consecutive task instances. In this thesis using the idea of introducing criticality levels from [29] we propose two methods for dealing with overload scheduling.

1.3 Outline of the report

The rest of the report is organized as follows.

- Chapter 2 provides background knowledge that is used throughout this thesis. This chapter after introducing the hierarchical scheduling framework provides a brief explanation of the control theory that is used in design of the controller.
- Chapter 3 describes design process of our AHSF in detail. In this chapter the budget controller is designed using an analytical approach. Then, two methods are suggested for handling overload situations in AHSF.
- Chapter 4 presents simulation results and examples. We have simulated HSF and added our budget controller in the simulation environment. In last section of this chapter we provide an example for illustrating introduced overload handling methods.
- Chapter 5 presents summary and conclusion of this thesis. Moreover, some possible trends of the thesis are suggested for the future work.
- In Appendix A a complete explanation of preparing simulation environment, adding control related functions and illustrating results are presented.

Chapter 2

Theoretical Background

2.1 The Hierarchical Scheduling Framework

In this thesis we investigate feedback scheduling in a single CPU where each CPU is modeled as a system S. Each system consists of a set of subsystems $S_S \in S$. The system is scheduled using a two level HSF as illustraterated in Figure 2.1. During run-time, the global scheduler chooses one of the subsystems and allocates CPU to that subsystem. Then, the subsystem's local scheduler shares this allocated CPU among its tasks according to its scheduling algorithm. As it is shown in Figure 2.1 we use a fixed priority algorithm in both local and global schedulers.

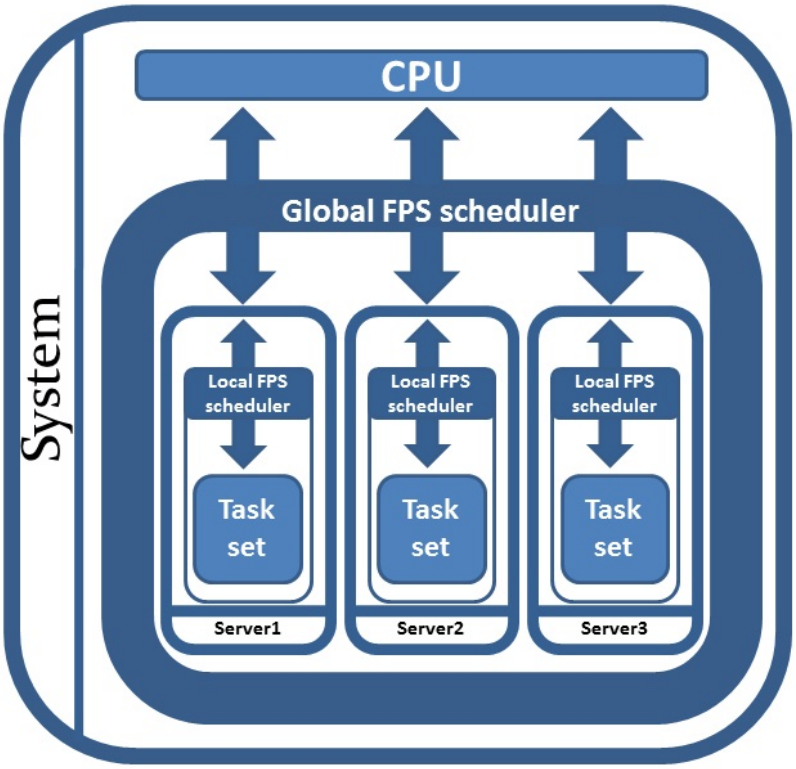

Figure 2.1: Hierarchical Scheduling Framework

2.1.1 Subsystem Model

Each subsystem S_S is represented by its timing interface parameters (T_S, P_S, B_S, ζ_S) where T_S, P_S, B_S and ζ are subsystem period, priority, budget and criticality respectively. Each subsystem S_S also consists of a set of tasks τ_S and a local scheduler. The criticality of subsystems ζ_S is used only in overload situations. Therefore, in modeling of not overloaded systems we ignore this parameter. In order to guarantee timing constraints of tasks we set subsystem period to half of its shortest task period. In each subsystem period T_S the subsystem budget B_S is reloaded. Budget of subsystems should be set to a minimum otherwise subsystems will have idle time and the resource will be wasted. Farhana et al. have proposed an algorithm for finding an exact minimum possible budget for subsystems that are using FPS in the local scheduler [16].

2.1.2 Task Model

We assume the periodic soft real-time task model $\tau_i(T_i, P_i, C_i, D_i)$, where T_i, P_i, C_i and D_i are task period, priority, worst-case execution time and relative deadline respectively. When a deadline miss happens in the system, the task continues executing until it finishes. Figure 2.2 shows a simple example that a task in a subsystem is scheduled using the received budget.

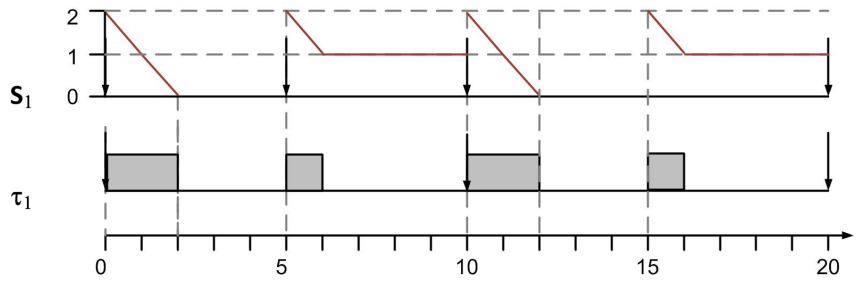

Figure 2.2: Example of scheduling one task inside on subsystem ($T_{S_1} = 5$, $B_{S_1} = 2$, $T_{\tau_1} = 10$, $C_{\tau_1} = 3$)

2.2 Feedback Control

In a closed loop system there is a plant which needs to be controlled. This control is done by frequently sampling the plant using some sensors and manipulating it according to the control logic. Plant manipulation is forwarded to the system using actuators. There are several controller design methods such as PID, LGQ, state feedback, etc. These controllers have different performance i.e., one may give better results than the others and also have different implementation complexity, however when designing any type of controller the mathematical model of the plant is needed. Figure 2.3 shows a computer control system, where the controller is implemented in a computer. Both output from plant and controller are discrete-time signals.

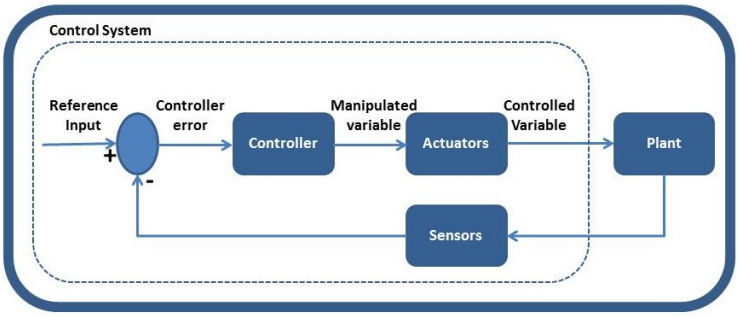

Figure 2.3: A computer controlled system

Since we use a simple PI (proportional-integral) controller in our adaptive framework, in this section basic information about PI controllers is provided. As it is mentioned in [26] the rationale behind not using the derivative term (D) is that this term might amplify noise when system load experiences significant changes.

2.2.1 PI Controller

Structure of a PI controller is shown in Figure 2.4. A PI controller consists of proportional and integral parts. Indeed, when we remove D (derivative) block from a PID controller we end up to a PI controller. At each sampling period, the controller samples the environment and calculates the error. The error is difference of the set point and current value of the controlled variable. Then, the manipulated variable is calculated using the following formula:

$$MV(t) = P_{out} + I_{out} \tag{2.1}$$

where $MV(t)$, P_{out} and I_{out} are the manipulated variable, output of the P and I block at time t respectively. Hence,

$$MV(t) = K_P\Delta(t) + K_I \int_{tw} \Delta(t)dt \tag{2.2}$$

where K_P, K_I, tw and Δ are proportional gain, integral gain, integral time window and current error respectively.

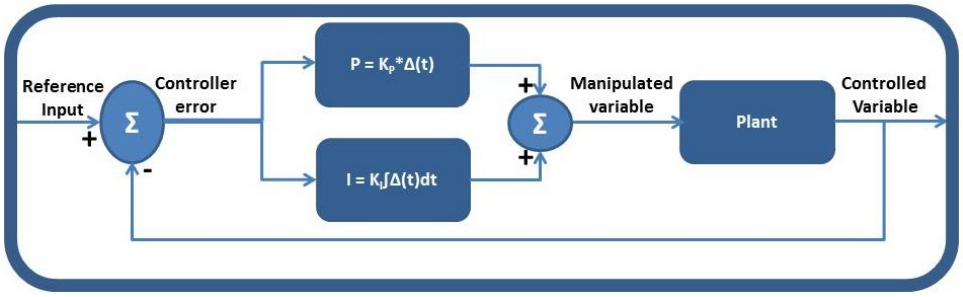

Figure 2.4: PI controller structure

Since we are dealing with a discreet system, we transform the control function to the z-domain using the z-transform. Model of the controller as well as model of the plant in z-domain is presented in Section 3.2.

Chapter 3

Design

3.1 Overview of The Adaptive Hierarchical Scheduling Framework

In our Adaptive Hierarchical Scheduling Framework (AHSF), each subsystem has one budget controller which is responsible for adapting the budget of the subsystem to its internal tasks demands. The budget controller finds a suitable budget value for its corresponding subsystem by periodically sampling the controlled variables. The subsystem does not receive the new budget unless it is approved by overload controller. The overload control logic only activated in overload situations. In normal mode, subsystems can acquire their necessary budget values. The architecture of our AHSF is illustrated in Figure 3.1.

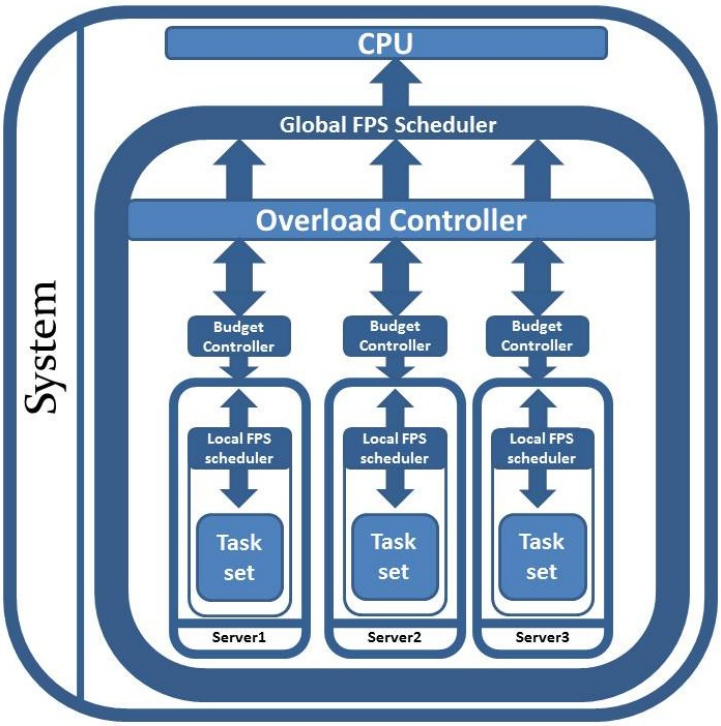

Figure 3.1: Adaptive Hierarchical Scheduling Framework

The budget controller uses two PI feedback loops for controlling the budget of subsystems. These loops are called "M-loop" and "U-loop" (see Section 3.2). While the "M-loop" tries to minimize the number of deadline misses, the "U-loop" keeps the budget of subsystems to a minimum possible value. Design process of the budget controller and the overload controller are described in next sections.

3.2 Budget Controller

The objective of this section is to provide detailed information about how control theory is applied to our HSF. Figure 2.4 shows a typical PI feedback control loop that will be used to control the budget of a subsystem. The controller changes the *manipulated variables* based on the input, which is the controller error, and the controller algorithm. The controller error is defined as the difference between *controlled variable* and the *reference input*. The first step in designing a controller is to define the controlled and manipulated variables, which are explained in this section. In the design of the controller we have used a similar approach as the one presented in [26]. The significance of our work is that we apply feedback control to the context of hierarchical scheduling.

We use two feedback loops to control the plant. The first loop is responsible for controlling the number of deadline misses and the second loop tries to reduce amount of idle time in the subsystems. The rationale behind using these two loops stems from basic principles of designing real-time systems. Since in a real-time system minimizing the number of deadline misses is of importance, we use the first loop for controlling deadline misses. On the other hand, it is desirable to keep system utilization close to 100% so that resources are not wasted. Hence, we use another loop for controlling idle time in the subsystems. To simplify the analyses, these feedback loops are considered to be independent from each other. Therefore, for each loop a set of controlled variables as well as analyses are presented seperately.

3.2.1 Controlled Variables

The first controlled variable is $M_S(t)$ which is defined as the total number of missed deadline jobs of all tasks inside the subsystem $\tau_i \in S_S$, within one specified time window (tw_m) prior to the current time t. The control loop which uses $M_S(t)$ as its controlled variable is called "M-loop" in the rest of the report. Architecture of the M-loop is shown in Figure 3.2.

The next controlled variable is $U_S(t)$ that is defined by the following formula

$$U_S(t) = \frac{B_S(t)}{E_S(t)} \tag{3.1}$$

where $B_S(t)$ and $E_S(t)$ represent total budget of the subsystem and total measured time of CPU usage by all tasks of the subsystem S_S in the time window tw_u respectively. Similar to the M-loop, we call the second control loop "U-loop" in the rest of the report. Architecture of the U-loop is shown in Figure 3.3.

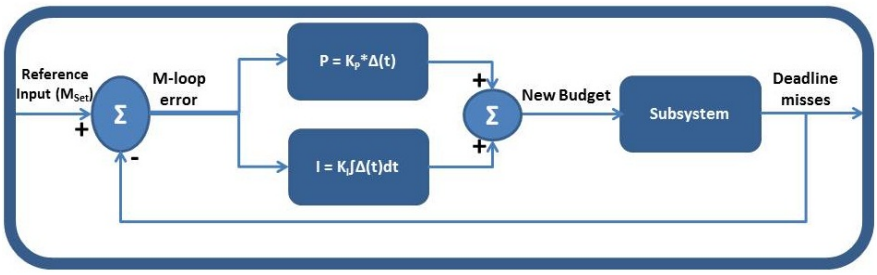

Figure 3.2: Architecture of the M-loop

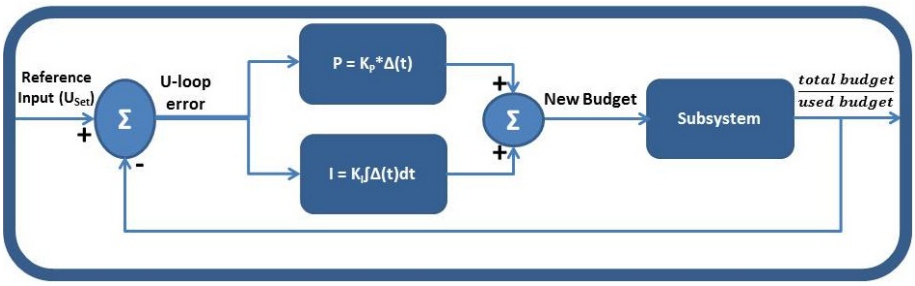

Figure 3.3: Architecture of the U-loop

3.2.2 Manipulated Variables

The budget of subsystem $B_S(t)$ is considered as the manipulated variable. The budget should be adjusted based on the error between the controlled variable and the reference input. In each sampling period, the controller adds budget change value $D_{B_S}(t)$ to the previous value of the subsystem budget.

$$B_S(t) = B_S(t-1) + D_{B_S}(t) \tag{3.2}$$

Figure 3.4 shows that how a PI controller is added to the architecture of our HSF for controlling the budget. This figure only illustrates the budget controller and the way it manipulates interface parameters of the subsystems (servers). It is important to notice that tunable parameters of the PI controllers are specific to each subsystem. Therefor we can assume there is a budget controller which corresponds to each subsystem, however, for simplification purpose we have illustrated only one control block in Figure 3.4.

3.2.3 Integrating loops

In the presented architecture, each feedback loop has a budget change output. We choose a budget change value which has greater absolute value (maximum operator). The reason for not using the minimum operation is that when loops are in their saturation zone, their output is zero. For example when the subsystem idle time is equal to the set point but there are some deadline misses, result of the U-loop is zero but result of the M-loop is a positive number. Hence, a logical operation for integrating results of the loops is the maximum operation.

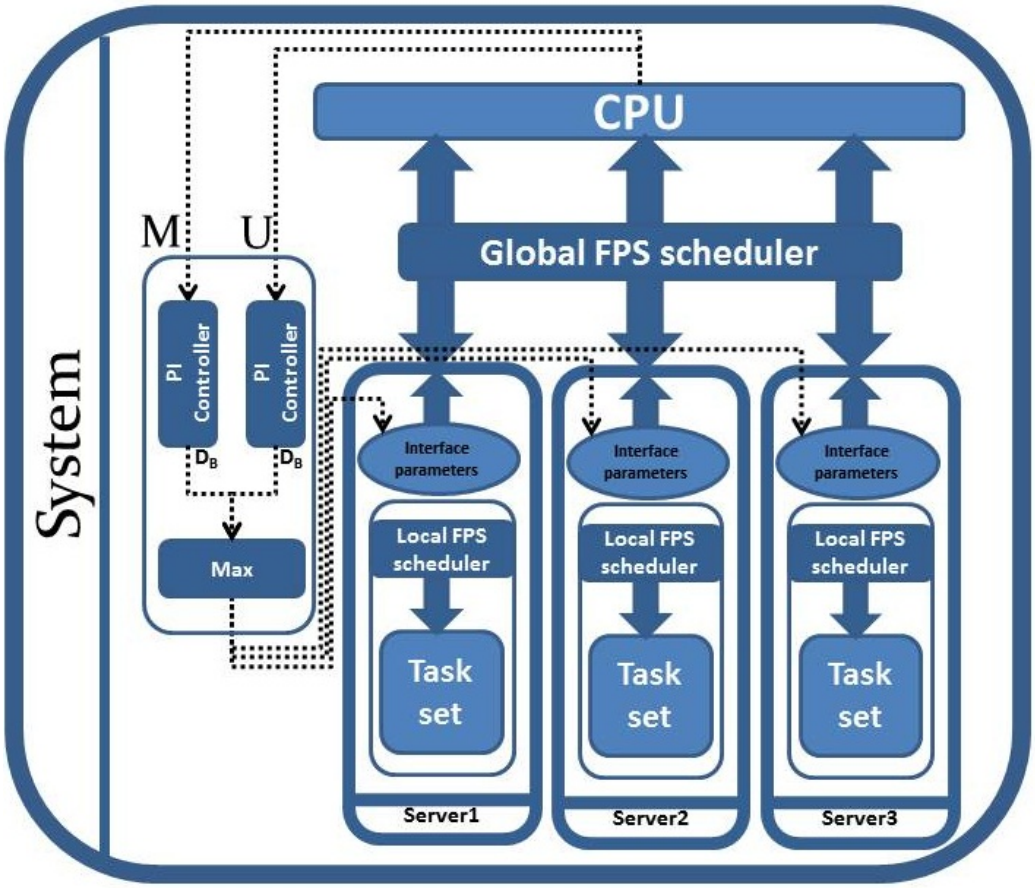

Figure 3.4: Architecture of our HSF with PI controllers

3.2.4 Model of Plant

In this section an approximate analytical model of the controlled system is presented. This model is useful when we are looking for optimal values of the tunable parameters in the controller. Based on the model and some analyses we find boundaries for tunable variables. Relation between the control output ($D_{B_S}(t)$) and controlled variables ($U_S(t)$ and $M_S(t)$) is of interest in the plant model. For the controlled variable $U_S(t)$, from the definition we have

$$U_S(t) = \frac{B_S(t)}{E_S(t)}. \tag{3.3}$$

In order to continue analysis we use $WCET_S = max(E_S(t))$. Hence

$$U_S(t) = \frac{B_S(t)}{WCET_S} \tag{3.4}$$

where $WCET_S$ is a total worst case execution time of all tasks in subsystem S_S. After transfering to the z domain, from the control input $D_B(z)$ to $U(z)$ the transfer function is:

$$U(z) = P_U(z)D_B(z) \tag{3.5}$$

and

$$P_U(z) = G_U/(z-1) \tag{3.6}$$

where

$$G_U = \frac{1}{WCET_S}. \tag{3.7}$$

We can define $M_{S_S}(t)$ based on U_{S_S} and derive a similar model for $M_{S_S}(t)$:

$$M_{S_S}(t) = M_{S_S}(t-1) + G_m(U_{S_S}(t) - U_{S_S}(t-1)) \tag{3.8}$$

where G_m is deadline miss factor and can be found by plotting the $M_{S_S}(t)$ curve as a function of $U_{S_S}(t)$. For continuing the analysis we use G_M as the maximum value of G_m. Similar to $U_{S_S}(t)$ we can derive the transfer function $P_M(z) = G_U G_M/(z-1)$.

3.2.5 Model of The Controller

A PI controller is used to control the plant. The PI controller function is

$$D_{B_S}(t) = K_P Erorr_S(t) + K_I \sum_{tw} Error_S(t) \tag{3.9}$$

where K_P, K_I, $Erorr_S(t)$ and tw are proportional gain, integral gain, error value of the subsystem S_S at time t and time window respectively. Gain variables are tunable parameters of the controller and should be tuned to get a desirable performance. Each control loop has its own controller. Therefore, introduced parameters are specific to each loop. $Erorr_S(t)$ is the difference between current value of the controlled variables and set points of the "M-loop" (M_{Set}) or set point of the "U-loop" (U_{Set}). After applying the z-transform we have

$$D_{B_S}(z) = K_P + \frac{K_I}{(z-1)}. \tag{3.10}$$

3.2.6 Closed-Loop System Model

If we consider $G = G_U G_M$ for the M-loop and $G = G_U$ for the U-loop, we can derive the following closed-loop system model for both loops:

$$H_S(z) = \frac{C(z)P(z)}{1 + C(z)P(z)} = \frac{GK_P(z-1) + K_I G}{(z-1)^2 + G(K_P(z-1) + K_I)} \tag{3.11}$$

3.2.7 Stability Analysis

From (3.11) the characteristic equation is:

$$(z-1)^2 + G(K_P(z-1) + K_I) = z^2 + \alpha_1 z + \alpha_2 \tag{3.12}$$

where $\alpha_1 = GK_P - 2$ and $\alpha_2 = 1 - GK_P + GK_I$. According to Jury's scheme [19, p. 82] the stability conditions are:

$$\alpha_2 < 1 \tag{3.13}$$

$$\alpha_2 > -1 + \alpha_1 \tag{3.14}$$

$$\alpha_2 > -1 - \alpha_1. \tag{3.15}$$

These conditions give us boundaries on tunable variables of the system.

3.2.8 Configurations

As it is shown in Figure 3.1, there is a budget controller corresponding to each subsystem. It means that controllers can be configured separately and they can have different periods, gain values and set points. Therefore, according to the requirements of tasks inside each subsystem its controller should be configured. As it is mentioned in Section 3.2.7, by applying stability analysis we make tuning process of gain variables (K_P and K_I) easy. This section provides discussions about tuning controller period and setting reference points in the feedback loops. However, we don't use any mathematical analysis; we provide a discussion about the rationale behind selecting the controller period and set points.

Tuning Controller Period

In most of the simulations presented in Chapter 4 the controller period is larger than the largest subsystem period multiplied by two. The reason for choosing such a period is to let all subsystems work with their new budget so that in the next sampling we can measure controlled variables that are result of the new budget settings. The problem of choosing a larger period is that "M-loop" cannot react to deadline misses quickly which is not desirable in some cases. A useful option that is possible in our AHSF is that we can set a relative short control period for the critical subsystems and a relative large period for the less critical ones. In summary, the controller period is an application specific tunable parameter which should be tuned according to the requirements of the subsystems.

Set Point of the M-loop

A desirable value for M is zero because in this case all subsystem tasks can meet their deadlines. However, since this value is on the threshold of deadline miss saturation, setting the reference point of M-loop to zero results to a very sensitive controller which in some cases conducive to a large steady state error. Therefore, in subsystems that can tolerate some deadline misses by choosing the M-loop reference point to one or two we can acquire a better performance.

Set Point of the U-loop

A desirable value for U is one because in this case subsystems are using all CPU portions that they have received. Similar to the M-loop, since $U = 1$ is a threshold value, setting the reference point of the U-loop to one is conducive to a large steady state error. Therefore a value between one and two is recommended for the set point. In most of the simulations presented in Chapter 4 we set the set point of the U-loops to 1.2.

3.3 Overload Controller

In an overload situation, which we will call a *critical mode*, the controller cannot provide all subsystems with enough budgets. Therefore, we suggest a mechanism for distributing the CPU among subsystems based on their criticality value ζ_S. In dealing with the critical mode, the very first issue is detecting the time which the system mode is changing from normal to critical. Furthermore, after providing the most critical subsystem with enough budget, other subsystems

should be able to use the remaining portion of the CPU resource such that they do not violate the schedulability condition of a higher criticality subsystem.

3.3.1 Mode Change

We suggest two mechanisms for detecting the mode change time. The goal of proposing these methods is to predict overload situations and to avoid critical deadline misses.

Both methods require conducting a schedulability analysis in global level. Since the global scheduler schedules subsystems in a similar way as scheduling simple real-time periodic tasks, it is possible to use the schedulability analysis methods used for scheduling periodic tasks. The subsystem can be modeled as a periodic task where the subsystem period is equivalent to the task period and the subsystem budget is equivalent to the task execution time.

Method one

According to offline analyses we can find a safe budget ceiling for all subsystems in which the whole system is schedulable and completely utilized. In doing so, we need to add an additional parameter to each subsystem in the subsystem interface which is the maximum budget value B_S^{Max}. Therefore, a system would be considered in its critical mode if any of its subsystems $S_S \in S$ is requesting a budget more than its maximum value ($B_S > B_S^{Max}$). It is important to note that changing the mode does not necessarily mean that there is no enough resources. We only force the system to do some additional checks by changing the mode.

In this approach, B_S^{Max} of the subsystems should be assigned to a safe value which could be a considerable safe boundary plus the time that all tasks inside that subsystem can finish their worst case execution time. This time duration can be calculated using the notion of real-time virtual processor model introduced by Mok et al. [28]. In calculating B_S^{Max} we should assume that all other higher criticality subsystems in the system $S_S \in S$ are using their B_S^{Max} and find a sufficient value for B_S^{Max} such that the response time of S_S is less than its period T_S.

Method two

Using an online schedulablity analysis in the global level, the system can detect the mode change time. Whenever the global scheduler fails to (analytically) schedule subsystems according to their new budget value, the system is considered to be in its critical mode.

3.3.2 Budget Distribution Policy in the Critical Mode

In the critical mode, the controller starts to share the budget among subsystems from the most critical subsystem to the least critical one. Therefore the most critical subsystem receives the entire budget that it requests. The amount of the budget that the lower criticality subsystems receive is completely dependent on the new budgets of the higher criticality subsystems. If a lower criticality subsystem asks for less than the maximum possible budget it will get it, otherwise it will get the maximum possible value. It is an undeniable fact that in the critical mode, less critical subsystems might completely be shut down or receive an small amount of the budget such that their tasks start missing their deadlines which is unavoidable. The important point to highlight here is that the criticality value of a subsystem should be assigned based on

criticality of its inner tasks. In the case that a subsystem is composed of mixed criticality tasks, one approach is to assign the average value of the tasks criticality to the subsystem criticality ζ_S and use a scheduler in the local level which takes the criticality level of tasks into account. In this approach subsystems should have a minimum budget value B_S^{Min} which indicates how much budget is necessary for only scheduling the high criticality tasks. Another approach is to simply assign the maximum criticality level of tasks to the subsystem criticality value. However, by using this approach the system discriminates between tasks that have the same criticality levels and belong to different subsystem criticality levels.

3.3.3 Calculating the Remaining Budget

As it is mentioned, in the critical mode after assigning the budget for a higher criticality subsystem, there would be limitation for the budget of a lower criticality subsystem. We present two approaches for mapping the consumed budget in a high criticality subsystem to the budget value of a low criticality subsystem. These approaches correspond to the methods presented for detecting the mode change.

Method one

If we are using an offline analysis for detecting a mode change, then we know B_S^{Max} for each subsystem, and we also know that if all subsystems use their B_S^{Max} the whole system is schedulable. In this method, after detecting the mode change, B_S^{Max} are assigned to their corresponding B_S.

For a system consisting of two subsystems S_i and S_j assuming $\zeta_i > \zeta_j$ and S_i requests a new budget that is α unit more than its maximum budget ($B_i = B_i^{Max} + \alpha$), the system enters to the critical mode and we initialize the budgets with the maximum budgets $B_S = B_S^{Max}$. In order to provide the high criticality subsystem with the requested budget we need to reduce the budget of the lower criticality subsystem. Hence, $B_j = B_j - \lceil \alpha \frac{T_j}{T_i} \rceil$. On the other hand, if afterwards S_i requests a budget value which is α unit less than its current budget B_i, we can transfer this extra budget to S_j using the following equation: $B_j = B_j + \lceil \alpha \frac{T_j}{T_i} \rceil$. These equations are used in implementation of the "TakeRequiredBudget(ζ_S, α)" and the "GiveExtraBudget(ζ_S, α)" functions presented in Algorithm 1 which shows pseudocode of method one. In this algorithm $NewBudget_i$ represents the new budget value that the budget controller suggests to the current subsystem S_i. When a subsystem requests a budget value which is less than its current budget we give the extra budget to the lower criticality subsystem using the "GiveExtraBudget($\zeta_i + 1$, α)" function. In the case that there is no other lower criticality subsystem in the system, this function reserves the extra budget in the lowest criticality subsystem such that the "TakeRequiredBudget(ζ_i, α)" function can use this spare budget.

The "TakeRequiredBudget(ζ_i, α)" function takes the required budget from some of the lower criticality subsystems (depending on amount of the requested budget) in such a way that S_i can receive the maximum available budget value. Indeed, when a subsystem asks for a budget value which is more than its current budget B_i, the controller takes this amount from the lowest criticality subsystem. If the lowest criticality subsystem cannot afford the whole required budget, the "TakeRequiredBudget" function gets the entire lower criticality subsystem budget and takes the remaining requested budget from the subsystem which belongs to the one level higher criticality (if its criticality is lower than criticality of the requested subsystem). The

main purpose of exchanging budgets among subsystems in this method is to keep track of the overall available budget.

Algorithm 1 Method one

```
for ζ_i = 0 to ζ_i = n − 1 do
    α = |B_i − NewBudget_i|;
    if NewBudget_i < B_i then
        GiveExtraBudget(ζ_i + 1, α);
        B_i = NewBudget_i;
    end if
    if NewBudget_i > B_i then
        B_i = B_i + TakeRequiredBudget(n − 1, α, i);
    end if
end for
```

Algorithm 2 shows implementation of the "GiveExtraBudget" function. In this function we check whether there exists called criticality level or not. If it does not exists we reserve the budget in a variable, otherwise we add the extra budget to the current subsystem budget.

Algorithm 2 GiveExtraBudget(ζ_i, α)

```
if ζ_i > n − 1 then
    reservedBudget = ⌈α T_i / T_{i−1}⌉;
else
    B_i = B_i + ⌈α T_i / T_{i−1}⌉;
end if
```

Algorithm 3 shows implementation of the "TakeRequiredBudget" function. In this function first we try to provide required budget from the reserved budget. If it is not possible we continue taking the required budget from the lowest criticality subsystem, and if it is not enough we move to one level higher criticality and claim the budget from that subsystem.

Method two

In this approach, we should do a schedulability analysis after each new budget assignment. In contrast with the global schedulability test which is done for mode change detection, in conducting the schedulability analysis for the subsystem S_i the algorithm assumes that all lower criticality subsystems are shut down. In doing so, when a higher criticality subsystem requires more budget, the algorithm punishes the lowest criticality subsystem. When the system is not schedulable, we have to rollback the last budget assignment and assign a lower value to the budget of that subsystem. Algorithm 4 shows pseudocode of method two. In this pseudocode the "$Schedulable(S, S_i)$" function conducts a schedulability analysis according to the new budget values. Furthermore, the $FindNewBudget(S_i, B_i)$ function returns a new value for the budget of subsystem S_i based on the last failed value.

Algorithm 3 TakeRequiredBudget($start$, α, end)

if $\lceil \alpha \frac{T_{start}}{T_{end}} \rceil \leq reservedBudget$ **then**
  $reservedBudget = reservedBudget - \lceil \alpha \frac{T_{start}}{T_{end}} \rceil$;
  return;
else
  $\alpha = \alpha - \lceil reservedBudget \frac{T_{start}}{T_{end}} \rceil$;
  $reservedBudget = 0$;
end if
for $\zeta_i = start$ downto $\zeta_i = end - 1$ **do**
  if $\lceil \alpha \frac{T_i}{T_{end}} \rceil \leq B_i$ **then**
    $B_i = B_i - \lceil \alpha \frac{T_i}{T_{end}} \rceil$;
    return;
  else
    $\alpha = \alpha - \lceil B_i \frac{T_i}{T_{end}} \rceil$;
    $B_i = 0$;
  end if
end for

Algorithm 4 Method two

for $\zeta_i = 0$ to $\zeta_i = n - 1$ **do**
  $B_i = NewBudget_i$;
  while $Schedulable(S, S_i) \neq True$ **do**
    $B_i = FindNewBudget(S_i, B_i)$;
  end while
end for

Chapter 4

Simulation and Examples

4.1 Simulation Results

The simulation environment is prepared by modeling the HSF in the Times tool and generating C++ files from the model [7]. The generated code is extended so that it contains the designed PI controller function. In addition, some functions are added for calculating the controlled variables in the scheduler body. A variety of simulation examples using different systems as well as different configurations are presented in this chapter. More information about preparing the simulation environment is presented in Appendix A.

4.1.1 Base Simulation

In this simulation example we present an example scenario that shows budget adaptation in situations which the execution time of a task in one subsystem varies from low to high and vice versa. We show how the budget is changed in response to the new load condition of the subsystem.

There are totally two subsystems in the system. In both global and local levels we use the fixed priority algorithm for scheduling subsystems and tasks. Specifications of subsystems are shown in Table 4.1. We assume that tasks which are inside S_1 have a fixed execution time and that using the pre-assigned budget they can meet their deadlines. In subsystem S_2 there are two tasks, and their specifications are shown in Table 4.2. We also assume that task one ($\tau 1$) experiences some changes in its execution time during run-time. Execution time changes are shown in Table 4.3. The execution time variation is done using a function which is responsible for changing execution time of tasks to a predefined value at a specific clock cycle. In the presented simulation, execution time is changed in a range such that it does not violate the whole system schedulability condition.

Name	T_S	B_S	P_S
S_1	19	2	1
S_2	5	3	0

Table 4.1: Subsystems specifications

Name	T_i	D_i	P_i	C_i
τ1	10	6	1	3
τ2	11	8	0	1

Table 4.2: Tasks specifications of S_1

Time	0	50	200	400
C_i	3	2	3	0

Table 4.3: Execution Time Changes of τ1

The *tw* and controller period are considered to be 15 in this example. Hence, every 15 ticks the controller measures the controlled variables, and based on their value takes action by changing the budget of the subsystem one S_1. The controller period is experimentally tuned by taking into consideration the trade-off between calculation overhead and the controller response speed (see Section 3.2.8). In order to have a faster reaction to the environment changes, we can decrease the controller period. Consequently, it can sample and actuate more frequently which however increases the run-time overhead. After changing the controller period, tunable variables of the controller should be tuned to acquire a better controller performance. The controller is implemented inside the scheduler such that the scheduler runs the controller function (periodically) before other parts of the code.

The system is executed for 600 ticks, and the controlled variables as well as the budget are sampled in each controller execution, and the result is illustrated in Figure 4.1. As it is shown in Figure 4.1, 15 ticks after the system starts execution, the controller observes one deadline miss. It means that the pre-assigned budget is not enough for the tasks of S_1 to meet their deadlines. After observing the deadline miss, the controller increases the budget and after that all subsystem tasks are able to finish execution before their corresponding deadline. At time 50, when the execution time of τ1 is reduced from three to two, the controller reduces the budget from four to two in two steps. At time 200, the execution time of τ1 is increased to three and it causes some deadline misses. After which the controller observes the deadline misses, it increases the budget. Finally, the last change happens at time 400, when the execution time of τ1 is reduced to zero. In this case the U-loop experiences a huge error value which is conducive to a sudden change of the budget from four to three and eventually to one.

The important point to highlight here is that when we move from low to high execution time, the M-loop plays an essential role in adapting the budget. On the other hand, when the execution time is decreased, the U-loop adapts the budget according to the current requirements of the system.

In order to illustrate the difference between having an adaptive budget and having a pre-assigned fixed budget, we have conducted a set of simulations using a fixed budget and we have measured the amount of idle time and the number of deadline misses of S_1 during first 600 ticks. Table 4.4 shows a comparison between using different budgets and using our adaptive approach. Since the scheduler does not support execution of a task after its period, we couldn't measure values for the budget equal to one.

Budget	3	2	1	adaptive
Deadline misses	12	33	-	4
Idle time	197	77	-	185

Table 4.4: Idle time and deadline misses using different budgets

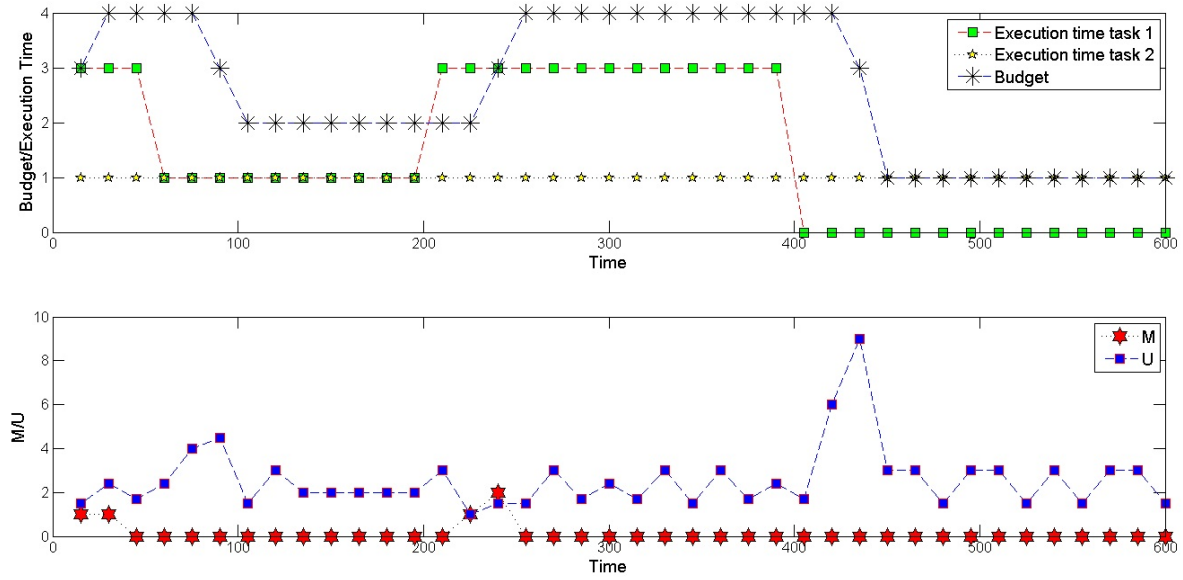

Figure 4.1: Execution times, budget and controlled variables change over time

4.1.2 Different Configurations

In this part we set up the same system as previous one but we change configuration and conduct some other simulations.

Control Period = 30

In this example, we are using the same system as previous one but the only difference is the controller period. We change the controller period to 30 and the system is executed again. As it is shown in Figure 4.2 we can see a similar budget adaptation pattern but with some delay. In addition, the number of deadline misses is increased to six which was four in the previous case.

Control Period = 10

In another example we set 10 to the controller period and execute the system. As it is shown in Figure 4.3 this time the number of deadline misses is decreased. Using this configuration system experiences two total deadline misses and the maximum number of deadline miss in one control period is one. The earlier controller sees a deadline miss, the faster it can react. When we are changing the controller period, we should consider the control overhead on the scheduler.

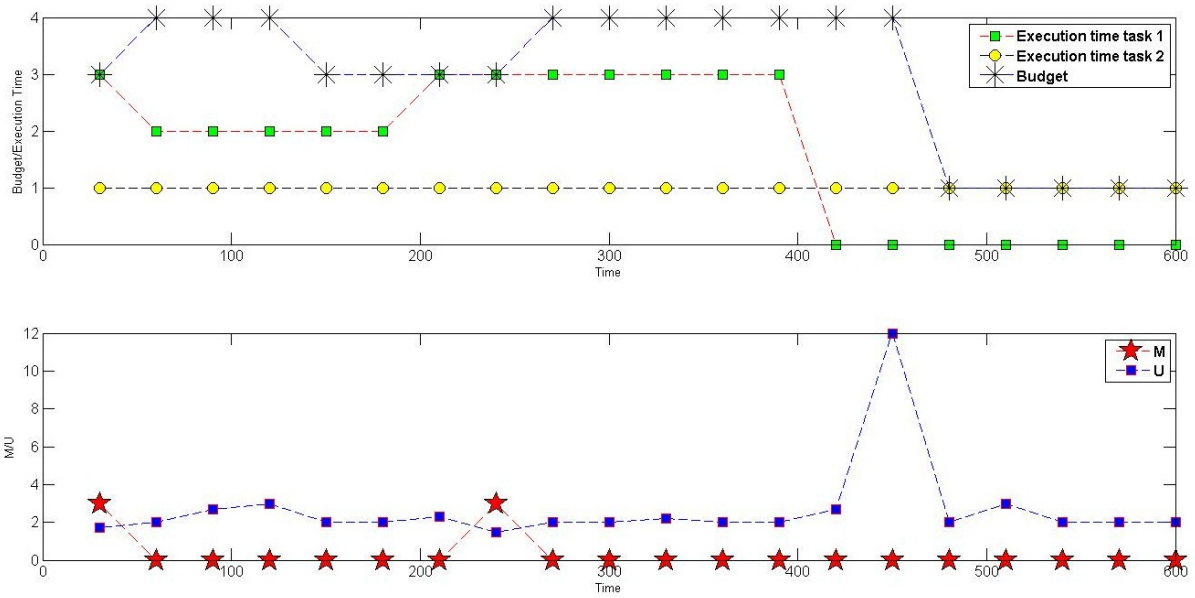

Figure 4.2: Execution times, budget and controlled variables change over time (Control period = 30)

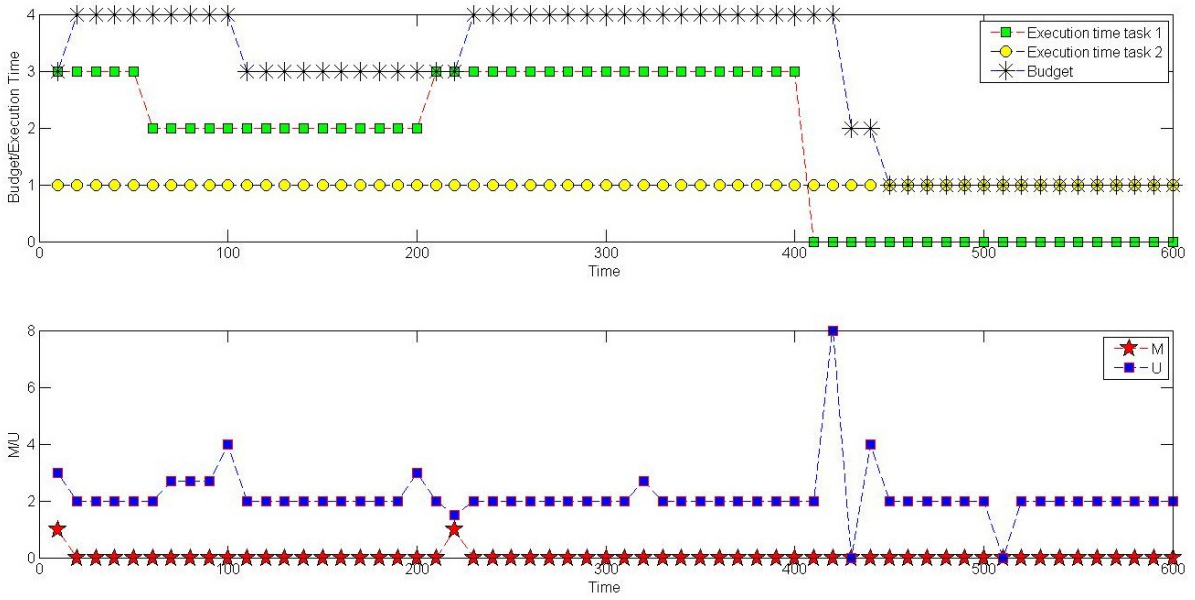

Figure 4.3: Execution times, budget and controlled variables change over time (Control period = 10)

Different Set points

In this example set point of the "M-loop" and "U-loop" are set to 0.5 and 1.2 respectively and result is shown in Figure 4.4. In the previous examples the set points were 0 and 1.5. Using this configuration system experiences 10 total deadline misses and the maximum number of deadline miss in one control period is two. On the other hand the total idle time is 134 which

is fewer than the base simulation. This example shows that when a system can tolerate some deadline misses, by configuring a value more than zero to the set point of "M-loop" and a value close to one to the set point of "U-loop" system can achieve a high utilization.

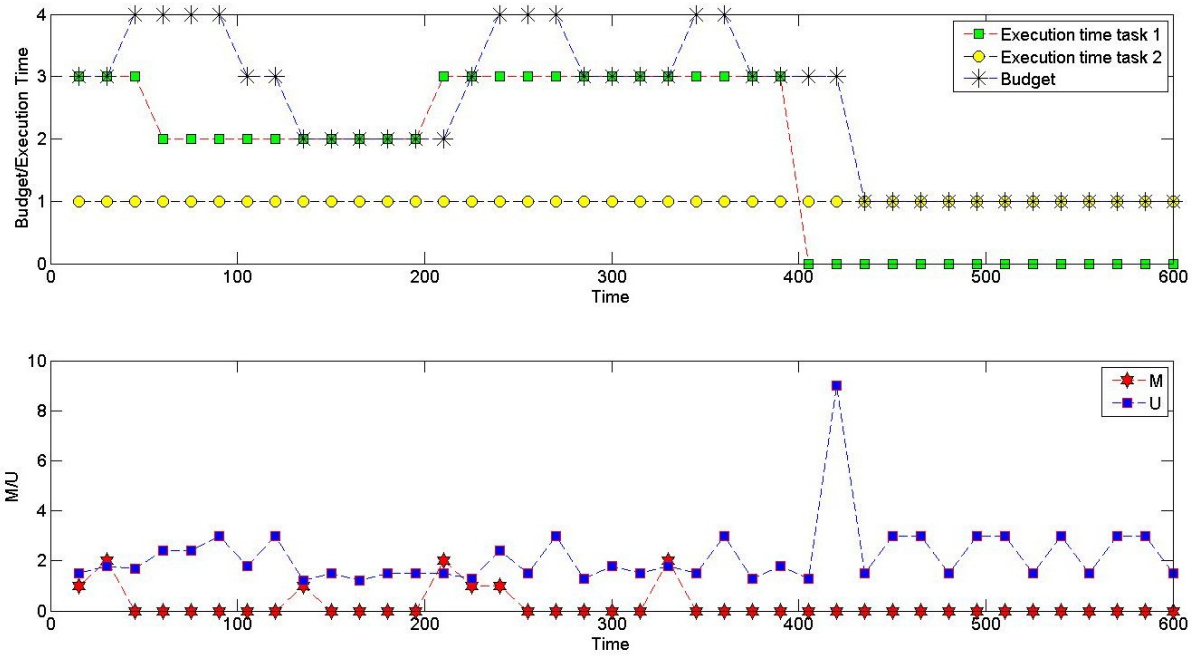

Figure 4.4: Execution times, budget and controlled variables change over time ($M_{Set} = 0.5$ and $U_{Set} = 1.2$)

4.1.3 Four Subsystems

Assume that we have four subsystems in a system. Table 4.5 shows specification of these subsystems. In order to show the performance of our AHSF, we scheduled this system and plotted the budget and controlled variables over time in Figure 4.5. The fixed priority scheduling algorithm is used in both local and global schedulers. Since execution time of tasks inside subsystems are changing over time, the budgets of subsystems are adapted during run-time to improve whole system performance. Figure 4.5 indicates the relation between controlled variables of the two feedback loops and the budget.

Name	T_S	Initial B_S	P_S	Number of tasks
S_1	23	1	1	4
S_2	20	2	0	2
S_3	20	2	2	2
S_4	31	1	3	1

Table 4.5: Subsystems specifications

Tasks specifications of all subsystems are presented in Table 4.6. Budget adaptation in this example is because of two reasons. First of all, initial budget values are not enough for some subsystems. In addition, execution time of some tasks change at time 200 and 300.

Subsystem	Name	T_i	D_i	P_i	C_i
S_1	$\tau 1$	48	30	0	1
S_1	$\tau 2$	48	40	1	1
S_2	$\tau 1$	40	32	0	1
S_2	$\tau 2$	40	30	1	1
S_3	$\tau 1$	40	35	0	1
S_3	$\tau 2$	42	40	1	1
S_3	$\tau 3$	44	34	2	1
S_3	$\tau 4$	48	19	3	0
S_4	$\tau 1$	60	30	0	2

Table 4.6: Tasks specifications of all subsystems

In the illustrated example execution time of tasks in subsystems vary in a range such that the whole system remains schedulable. Therefore, the overload controller is not active and Figure 4.5 only shows performance of the budget controllers. The presented overload control techniques in Section 3.3 will be implemented in the same prepared simulation environment.

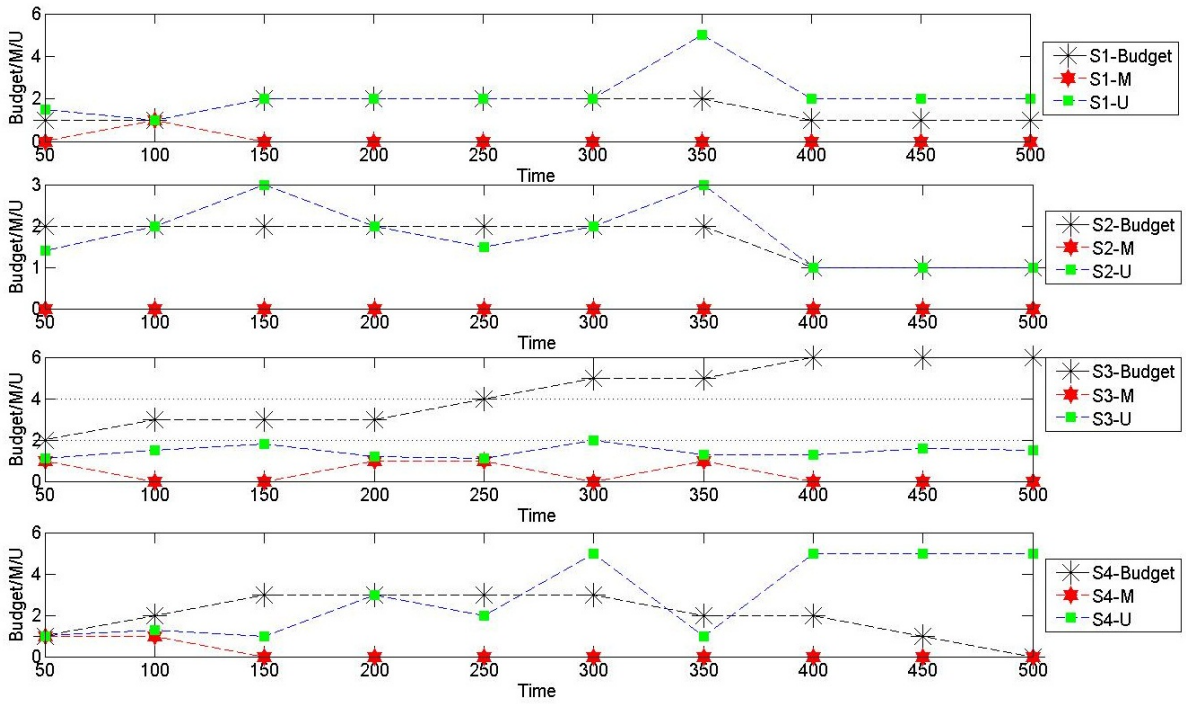

Figure 4.5: Budget adaptation of four subsystems over time (S_i-M and S_i-U are controlled variables of the "M-loop" and "U-loop" respectively.)

4.2 Overload Control Example

Assume a system with the specifications presented in Table 4.7. In order to illustrate the introduced approaches, we present a scenario and show how we can apply these two methods to schedule the example system in the critical mode.

Name	T_S	Initial B_S	B_S^{Max}	P_S	ζ_S
S_1	20	1	2	1	1
S_2	22	3	4	2	2
S_3	18	2	2	0 (highest)	3
S_4	19	8	8	3	0 (highest)

Table 4.7: Subsystems specifications

Assume that S_1 has current budget $B_1 = 2$ and that it requires two additional units of budget, and also assume that this will cause a mode change from normal to critical mode. The other subsystems have their initial budget values and they want to keep their budgets unchanged. If we want to schedule the system without considering their criticality, the timing constraints of S_1 are guaranteed only if the remaining budget after using S_3 is sufficient for S_1. However, since $\zeta_1 > \zeta_3$, in interfering of S_1 by S_3 we want S_1 to use the CPU.

4.2.1 Method one

By using method one, the following events will happen after the request of S_1:

- Since $B_1 > B_1^{Max}$ the system mode will change to the critical mode and $B_S = B_S^{Max}$ ($B_1 = 2$, $B_2 = 4$, $B_3 = 2$ and $B_4 = 8$).
- $B_4 = 8$ and S_4 asks for 8 budget units. Therefore, the budget of S_4 remains unchanged.
- TakeRequiredBudget(3, 2, 1) will take two units of budget from S_3 and will give it to S_1. Hence, $B_1 = 4$ and $B_3 = 0$.
- Since S_2 asks for a budget less than its current budget, the "GiveExtraBudget(3, 1)" will change $B_3 = 1$ and $B_2 = 3$.
- $B_3 = 1$ and S_3 asks for 2. Since there is no other lower criticality subsystem and no reserved budget the "TakeRequiredBudget" function returns 0 and B_3 remains unchanged.
- After the overload controller finishes its job, the budget values will be: $B_1 = 4$, $B_2 = 3$, $B_3 = 1$ and $B_4 = 8$.

4.2.2 Method two

By using method two, the following events will happen after request of S_1:

- The global schedulability check will fail (assumption) and it will cause a mode change.
- The budget of S_4 will change $B_4 = 8$ and the schedulability analysis assuming $B_1 = B_2 = B_3 = 0$ will be successfully done.
- The budget of S_1 will change $B_1 = 4$ and the schedulability analysis assuming $B_2 = B_3 = 0$ will be successfully done.
- The budget of S_2 will be set $B_2 = 3$ and a schedulability test assuming $B_3 = 0$ will be done. Since the system is schedulable the algorithm will move to the next step.
- The budget of S_3 will be set $B_3 = 2$ and a schedulability test will be done. Since the system is not schedulable it will assign $B_3 = 1$ and perform the schedulability test again. Since this time the system is schedulable the algorithm will move to the next step.

4.3 Discussion

In this section we have investigated different systems with different parameters. Simulation results show that a relative short controller period is desirable in systems that cannot tolerate deadline misses. In such a system we can reduce number of deadline misses by imposing more overhead to the scheduler. On the other hand, in the systems that high utilization is the most important issue, we can assign set points of the M and U control loops to a value more than zero and a value around 1.5 respectively.

Introduced overload methods seems to have a similar performance. However, the offline method seems to have slightly lower overhead because some parts of the calculations are done offline.

Chapter 5

Summary and Future Work

5.1 Summary and Conclusions

In this thesis we have used feedback control techniques in the context of a hierarchical scheduling framework for adapting the budgets of subsystems during run-time. Using a mathematical modeling approach, we have designed a PI controller which by sampling the number of deadlines and the duration of idle time in subsystems takes an action and manipulates the budget.

Simulation results show that the controller is able to adapt the budget when execution times of tasks are changed. When the system is not overloaded, manipulating the budget of one subsystem either lets more subsystem tasks to meet their deadlines or it decreases the response time of the tasks that are inside the other subsystems in the same node. For simulation examples, we have designed a scenario in which subsystem task experiences both execution time increase and decrease. We have investigated different control frequencies and set points on the same example scenario and results are shown in Chapter 4. Finally, a system consisting of four subsystems is simulated in which some subsystems are demanding more budget and others do not use their budget. The controller successfully adapted the budget according to subsystem demands.

In addition, we propose one online and one offline method for controlling the budget adaptation in the critical mode. Although we have not implemented the overload controller in our simulation environment, using an example we have illustrated our overload control methods. When it comes to implementation, different approaches can be used for implementing these methods. For instance, the schedulability test can be done using either the utilization based test, the response time analysis or an approximation approach such as the one presented in [17].

In compositional real-time systems where execution time of tasks are not fixed in run-time or even in systems where task execution times are not known before run-time, our adaptive hierarchical scheduling framework provides the system developer with an applicable solution for scheduling real-time tasks. Our AHSF makes it possible to adaptively schedule mixed criticality subsystems on a single node by allowing different configurations for subsystems budget controllers.

5.2 Future Works

The work presented in this thesis can be extended in different aspects.

1. Multi-mode real-time systems [31] can be investigated and integrated with hierarchical scheduling. When a system is experiencing a mode shift, interface variables can be adapted in a similar approach as we do in our AHSF.

2. Situations that by changing the budget of a subsystem, the system becomes not schedulable can be studied using our simulation environment. We have suggested two methods in this thesis which can be studied using simulation examples.

3. A set of experiments for comparing the response time of tasks in AHSF and HSF can be conducted. Benefit of using AHSF can be judged by comparing response time of tasks in HSF and AHSF.

4. Overhead of the controller can be investigated by implementing the presented controller and conducting some experiments on hardware. Overhead of the scheduler with controller and scheduler without the controller can be compared. Then, we can realize how much the cost of having an adaptive framework is.

5. The presented budget control approach can be investigated on hierarchical scheduling frameworks that have more than two levels.

6. We have studied systems with fixed priority algorithms in both local and global levels. Investigating other algorithm such as EDF in schedulers is another option for future works.

7. Investigating and applying feedback techniques in multicore HSFs is another trend for the future works.

8. Finally, another trend of our work is to study other types of controllers instead of the PI budget controller and investigate pros and cons of using different types of controllers in the context of adaptive hierarchical scheduling.

References

[1] Luca Abeni and Giorgio Buttazzo. Hierarchical QoS management for time sensitive applications. In *Proceedings of the 7th Real-Time Technology and Applications Symposium (RTAS '01)*, pages 63 –72, May 2001.

[2] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS '02)*, pages 71–80, December 2002.

[3] Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT '04)*, pages 95–103, September 2004.

[4] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '02)*, pages 460–464, April 2002.

[5] Bo Lincoln Johan Eker Karl-Erik Arzen Anton Cervin, Dan Henriksson. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *Control Systems Magazine*, 23:16–30, 2003.

[6] Mikael Åsberg, Thomas Nolte, Clara M. Otero Perez, and Shinpei Kato. Execution time monitoring in linux. In *Proceedings of the Work-In-Progress (WIP) session of 14th IEEE International Conference on Emerging Techonologies and Factory Automation (ETFA'09)*, September 2009.

[7] Mikael Åsberg, Paul Pettersson, and Thomas Nolte. Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling. In *Proceedings of the 23rd EUROMICRO Conference on Real-Time Systems (ECRTS'11)*, March 2011.

[8] S.R. Biyabani, J.A. Stankovic, and K. Ramamritham. The integration of deadline and criticalness in hard real-time scheduling. In *Proceedings of 7th the IEEE Real-Time Systems Symposium (RTSS '88)*, pages 152 –160, December 1988.

[9] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*, pages 286 – 295, December 1998.

[10] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, pages 90–99, December 1995.

[11] A. Cervin and J. Eker. Feedback scheduling of control tasks. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 5, pages 4871 –4876 vol.5, December 2000.

[12] T. Cucinotta, L. Palopoli, and L. Marzario. Stochastic feedback-based control of QoS in soft real-time systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 3533 – 3538 Vol.4, December 2004.

[13] Tommaso Cucinotta and Luigi Palopoli. Feedback scheduling for pipelines of tasks. In *Proceedings of the 10th international conference on Hybrid systems: computation and control (HSCC'07)*, pages 131–144, April 2007.

[14] G. de A Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *Computers, IEEE Transactions on*, pages 1332 – 1346, October 2003.

[15] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 308 –319, December 1997.

[16] Farhana Dewan and Nathan Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. Technical Report, Department of Computer Science Wayne State University Detroit, MI USA, March 2011.

[17] Nathan Fisher and Sanjoy Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. *Journal of Embedded Computing*, 2:291–299, December 2006.

[18] D. Henriksson, A. Cervin, J. Akesson, and K.-E. Arzen. Feedback scheduling of model predictive controllers. In *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '02)*, pages 207 – 216, September 2002.

[19] Bjorn Wittenmark Karl Johan Astrom. *Computer-Controlled Systems: Theory and Design (3rd Edition)*. Prentice Hall, 1996.

[20] Tei-Wei Kuo and Ching-Hui Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, December 1999.

[21] G. Lipari and S. Baruah. A hierarchical extension to the constant bandwidth server framework. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTAS '01)*, pages 26 –35, May 2001.

[22] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments. In *Proceedings of the 21st IEEE Real-time Systems Symposium (RTSS'00)*, pages 217–226, November 2000.

[23] Giuseppe Lipari and Sanjoy K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the 6th IEEE Real Time Technology and Applications Symposium (RTAS '00)*, pages 166 –175, May 2000.

[24] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTSŠ03)*, pages 151–158, July 2003.

[25] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*, pages 56 –67, December 1999.

[26] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, pages 85–126, 2002.

[27] Pedro Mejía-Alvarez, Rami Melhem, and Daniel Mossé. An incremental approach to scheduling during overloads in real-time systems. In *Proceedings of the 21st IEEE Real-time Systems Symposium (RTSS'00)*, pages 283–293, November 2000.

[28] A.K. Mok, X. Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings of the 7th Real-Time Technology and Applications Symposium (RTAS '01)*, pages 75–84, May 2001.

[29] Dionisio de Niz, Karthik Lakshmanan, and Ragunathan Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS '09)*, pages 291–300, December 2009.

[30] Thomas Nolte, Moris Behnam, Mikael Åsberg, Reinder J. Bril, and Insik Shin. Hierarchical scheduling of complex embedded real-time systems. In *Ecole d'Ete Temps-Reel (ETR'09)*, pages 129–142, August 2009.

[31] Linh T. X. Phan, Insup Lee, and Oleg Sokolsky. Compositional analysis of multi-mode systems. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS '10)*, pages 197–206, July 2010.

[32] P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *Parallel and Distributed Systems, IEEE Transactions on*, 10(6):549 –559, June 1999.

[33] S. Ramos-Thuel and J.K. Strosnider. The transient server approach to scheduling time-critical recovery operations. In *Proceedings of the 12th IEEE Real-Time Systems Symposium (RTSS '91)*, pages 286 –295, December 1991.

[34] S.G. Robertz, D. Henriksson, and A. Cervin. Memory-aware feedback scheduling of control tasks. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA '06)*, pages 70 –77, September 2006.

[35] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium, (RTSS '03)*, pages 2 – 13, 2003.

[36] John A. Stankovic, Tian He, Tarek Abdelzaher, Mike Marley, Gang Tao, Sang Son, and Cenyan Lu. Feedback control scheduling in distributed real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, pages 59 – 70, December 2001.

[37] Jianguo Yao, Xue Liu, Zonghua Gu, Xiaorui Wang, and Jian Li. Online adaptive utilization control for real-time embedded multiprocessor systems. *Journal of Systems Architecture*, pages 463 – 473, 2010.

[38] Fengxiang Zhang and Alan Burns. Analysis of hierarchical edf pre-emptive scheduling. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07)*, pages 423–434, December 2007.

Chapter 6

Appendix A

6.1 Tools Used for Simulations

In this appendix we explain tools that are used for simulating the proposed feedback control technique. In early phases of the thesis we investigated TrueTime simulation tool [5] and tried to conduct our simulation using this tool. Unfortunately, since the hierarchical scheduling is not supported by this tool we couldn't use it. Otherwise it is a powerful simulation tool for simulating control task.

6.1.1 Modeling HSF in Times

Modeling real-time schedulers in Times[1] tool is a useful approach which is proposed by Mikael Åsberg [7]. During this thesis different sample models that were available from his work are used for simulating the budget controller performance. Figure 6.1 shows the global scheduler automata that are modeled in Times tool. This example is used in four subsystems simulation that is presented in Chapter 4.

Using the Times tool we can generate C++ code from the automata model. In order to add the designed budget controller, we used this code synthesis tool which is provided in Times tool. Among available options in Times tool code generator we choose simulated kernel. Unfortunately, the generated code is not functioning properly and there are some bugs in it. Therefor, a C#.Net application is developed for fixing the bugs in generated C++ files.

6.1.2 Fixing the C++ Files

Times tool generates 12 C++ files from the model. From these 12 files four files are buggy and need to be fixed. Using the C#.Net application we should select "HierSched.c" file and then the program will look for other three files in the same folder. In addition XML file which contains the model should be in the same path. The application fixes files using information provided in the XML file and replacing some expressions. A new folder will be created and all necessary

[1]Times is a tool for modeling and implementation of embedded systems [4]. Since Times supports *task automata* (*timed automata with tasks*), it is used for modeling, verification and code synthesis purposes.

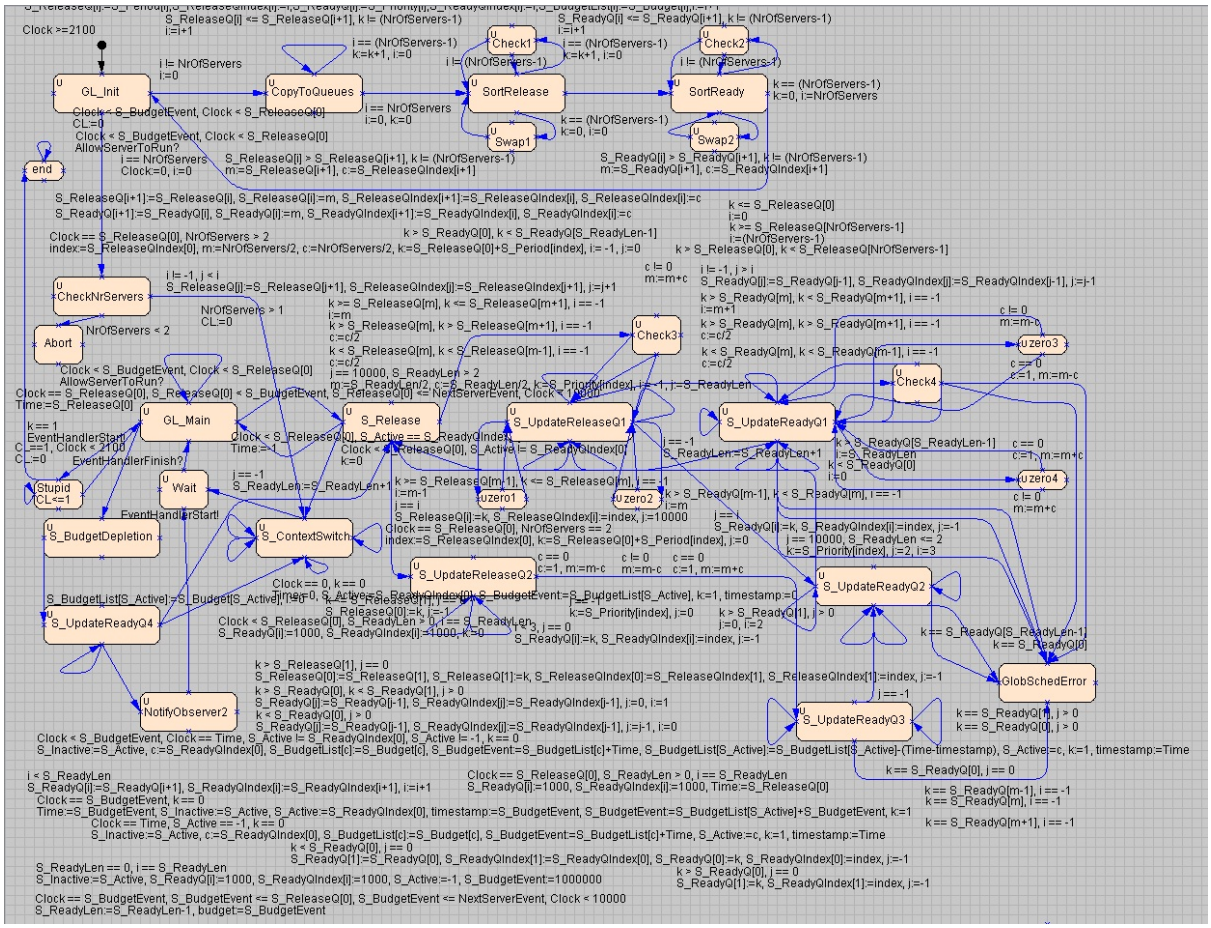

Figure 6.1: Global scheduler automata in Times tool

files would be in that folder. Then we need to compile and run "noos_kernel.c". Figure 6.2 shows a screen shot from the "C++ code fixer" application.

6.1.3 Adding The Control Code to Files

Among all generated files, two files need to be changed for adding the controller. The first file is "HierSched.c" which we should add some definitions and some functions. Following code is part of the HierSched.c file which contains control related declarations and functions. This code is used in the four subsystems simulation example in Section 4.1.3.

```
1  int S_deadlineMiss[NrOfServers];
2  int S_idleTime[NrOfServers];
3  int idleTime;
4  int totalBudget[NrOfServers] ={1,3,6,2};
5  int tmp       = 0;
6  char res      = 0;
7  int S1_Deadline[Server1_S1_NrOfTasks] = {30,40,1000};
8  int S2_Deadline[Server2_S2_NrOfTasks] = {32,30,1000};
9  int S3_Deadline[Server3_S3_NrOfTasks] = {35,40,34,19,1000};
10 int S4_Deadline[Server4_S4_NrOfTasks] = {30,1000};
```

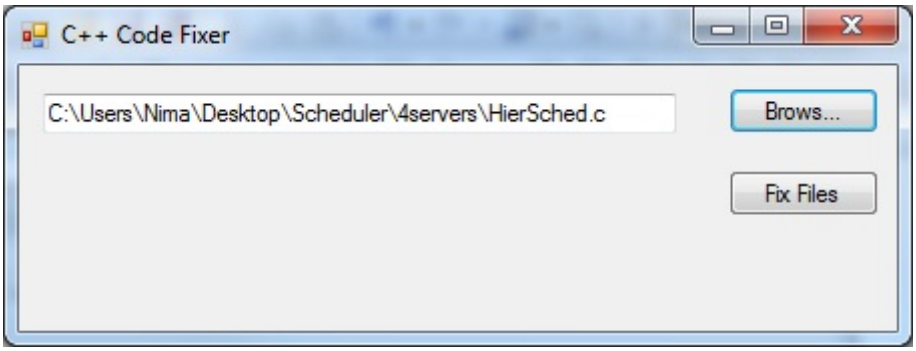

Figure 6.2: Screen shot of C#.Net application

```
11 //Functions
12 float UFunction( int i );
13 void ChangeExecTime( int i );
14 void Print( char input[], int i );
15 float PI(float ref, float value, float KP, float KI, float *PreviousError);
16 void Controller( void );
17 void serverRelease()
18 {
19     //Reloding the budget
20     totalBudget[S_ReleaseQIndex[0]] = totalBudget[S_ReleaseQIndex[0]] +
          S_Budget[S_ReleaseQIndex[0]];
21 }
22 void taskFinish(int i)
23 {
24      //Called in task finish transition
25      int activeTask;
26      switch(i)
27      {
28          case 0:
29               tmp = S1_Deadline[Server1_S1_Active];
30               activeTask = Server1_S1_Active+1;
31               S1_Deadline[Server1_S1_Active] += Server1_S1_Period[
                     Server1_S1_Active];
32               break;
33          case 1:
34               tmp = S2_Deadline[Server2_S2_Active];
35               activeTask = Server2_S2_Active+1;
36               S2_Deadline[Server2_S2_Active] += Server2_S2_Period[
                     Server2_S2_Active];
37               break;
38          case 2:
39               tmp = S3_Deadline[Server3_S3_Active];
40               activeTask = Server3_S3_Active+1;
41               S3_Deadline[Server3_S3_Active] += Server3_S3_Period[
                     Server3_S3_Active];
42               break;
43         case 3:
44               tmp = S4_Deadline[Server4_S4_Active];
45               activeTask = Server4_S4_Active+1;
46               S4_Deadline[Server4_S4_Active] += Server4_S4_Period[
                     Server4_S4_Active];
```

```
47                 break;
48         }
49         if(tmp > (int)sys_time)
50                 res = 'N' ;
51         else
52         {
53              res = 'Y';
54              S_deadlineMiss[i]++;
55         }
56 }
57 void ChangeExecTime(int i)
58 {
59          //Used for simulating execution time change of tasks
60          if(sys_time >= 100 && sys_time < 150 )
61          {
62             Server3_S3_ExecTime[1] = 0;
63             Server1_S1_ExecTime[0] = 5;
64             Server2_S2_ExecTime[0] = 3;
65          }
66          if(sys_time >= 200 && sys_time < 250)
67          {
68             Server2_S2_ExecTime[1] = 5;
69             Server3_S3_ExecTime[2] = 0;
70             Server3_S3_ExecTime[0] = 0;
71          }
72
73          if(sys_time >= 300 && sys_time < 350)
74          {
75             Server1_S1_ExecTime[1] = 2;
76          }
77          return;
78 }
```

These declared functions should be called from appropriate places in code. For example taskFinish function is called from transitions (code lines that are corresponding to automata model transitions) that a subsystem task is finished. For instance the following code shows taskFinish function call of subsystem S_3. This code is used in the four subsystems simulation example in Section 4.1.3.

```
1   case 143:
2     Server3_S3_ExecList[Server3_S3_Active]=Server3_S3_ExecTime[
          Server3_S3_Active];
3     Server3_i=0;
4     taskFinish(2);    //Call to taskFinish function from subsystem 3
5     break;
```

The second file which we should manipulate is "noos_hw_sim.c". This file contains implementation of the PI controller and budget adaptation. Following code should be added to noos_hw_sim.c. Following code is part of the noos_hw_sim.c that is used in the four subsystems simulation example in Section 4.1.3.

```
1 //Declerations
2 float Error, U_PreviousError, M_PreviousError, Budget, U_KP = 0.3, U_KI =
      0.1, M_KP = 0.7, M_KI = 0.2;
3
```

```
4  float rndup(float n)
5  {
6    //Used for budget adaptation
7    float t;
8    t=n-floor(n);
9    if (t>=0.5)
10     n = ceil(n);
11   else
12     n = floor(n);
13   return n;
14 }
15 float PI(float ref, float value, float KP, float KI, float *PreviousError)
16 {
17   //Implementation of the PI controller
18   float D;
19   Error = ref - value;
20   D = Error * KP + *PreviousError * KI;
21   *PreviousError = Error;
22   return D;
23 }
24 void ResetServerIdleTime(int i)
25 {
26   //Resets idle time of server i
27   int defaultIdle = 1000;
28   switch(i)
29   {
30     case 0:
31       Server1_S1_ExecList[Server1_S1_NrOfTasks-1] = defaultIdle;break;
32     case 1:
33       Server2_S2_ExecList[Server2_S2_NrOfTasks-1] = defaultIdle;break;
34     case 2:
35       Server3_S3_ExecList[Server3_S3_NrOfTasks-1] = defaultIdle;break;
36     case 3:
37       Server4_S4_ExecList[Server4_S4_NrOfTasks-1] = defaultIdle;break;
38   }
39   return;
40
41 }
42 void Controller()
43 {
44    //This function is called every control period
45    float D1, D2;
46    int i;
47    for(i=0;i<NrOfServers;i++)
48    {
49      Budget = S_Budget[i];
50      D1 = PI(1.5, UFunction(i), U_KP, U_KI, &U_PreviousError);
51      D2 = -PI(0, S_deadlineMiss[i], M_KP, M_KI, &M_PreviousError);
52      if(fabs(D1) > fabs(D2))
53         Budget += D1;
54      else
55        Budget += D2;
56      Print("Control", i);
57      Budget =  rndup(Budget);
```

```
58      //limit budget so it does not violate schedulablity condition of the
            other subsystems
59      if(Budget > 6 )
60          Budget = 6;
61      if(Budget < 1  )
62         Budget = 0;
63       //Disable Controller
64      //Clear history
65      totalBudget[i] = (S_Budget[i] - GlobalScheduler_S_BudgetList[i]);
66      ResetServerIdleTime(i);
67      S_deadlineMiss[i] = 0;
68
69      //Set new budget
70      S_Budget[i] = Budget;
71    }
72      return;
73 }
74 int ServerIdleTime(int i)
75 {
76   //returns idel time of server i
77   int idle;
78   switch(i)
79   {
80     case 0:
81       idle = Server1_S1_ExecList[Server1_S1_NrOfTasks-1];break;
82     case 1:
83       idle = Server2_S2_ExecList[Server2_S2_NrOfTasks-1];break;
84     case 2:
85       idle = Server3_S3_ExecList[Server3_S3_NrOfTasks-1];break;
86     case 3:
87       idle = Server4_S4_ExecList[Server4_S4_NrOfTasks-1];break;
88   }
89   return 1000 - idle;
90
91 }
92 float UFunction(int i)
93 {
94     //Calculates U (The controlled variable)
95     float U;
96     int localTotalBudget = totalBudget[i] - (S_Budget[i] -
           GlobalScheduler_S_BudgetList[i]);
97     if(localTotalBudget < ServerIdleTime(i))
98     localTotalBudget = ServerIdleTime(i);//because of late update of
           execList (after task switch)
99     if((float)(localTotalBudget - ServerIdleTime(i)) == 0)
100    {
101        return 5;
102    }
103    U = (float)localTotalBudget/(float)(localTotalBudget - ServerIdleTime(i
           ));
104    return U;
105 }
106 void Print(char input[], int i)
107 {
108   //used for plotting results
```

```
109   printf("S%d-%s Budget= %d M= %d idle= %d totalB= %d U= %3.1f E1= %d E2= %
          d %d\n", i+1, input, S_Budget[i], S_deadlineMiss[i], ServerIdleTime(i)
          ,      totalBudget[i] - (S_Budget[i] - GlobalScheduler_S_BudgetList[i
          ]) , UFunction(i), Server4_S4_ExecTime[0], Server2_S2_ExecTime[1], (
          int)sys_time);
110 }
```

6.1.4 Plotting The Results

In each control period, before adapting the budgets we call Print function which prints all controlled variables, current budgets and execution time of some tasks to screen. Then we redirect these outputs to a text file and using a MATLAB script we plot results. The following code is the MATLAB script that is used in the four subsystems simulation example in Section 4.1.3.

```
1  function [Inputs, Output] = FetchData()
2  j= 1;
3  [Server B Budget D M I Idle totalB TB T U E1 Exec1 E2 Exec2 Time] =
       textread('output.txt','%s %s %d %s %d %s %d %s %d %s %f %s %d %s %d %d')
       ;
4  s = sum(Idle)
5  s2 = sum(M)
6  S1_j = 1;
7  S2_j = 1;
8  S3_j = 1;
9  S4_j = 1;
10 for i=1:length(Server)
11     temp = (Server{i});
12     switch temp
13         case 'S1-Control'
14             S1_Budget(S1_j) = Budget(i);
15             S1_M(S1_j) = M(i);
16             S1_U(S1_j) = U(i);
17             S1_Time(S1_j) = Time(i);
18             S1_j = S1_j +1;
19         case 'S2-Control'
20             S2_Budget(S2_j) = Budget(i);
21             S2_M(S2_j) = M(i);
22             S2_U(S2_j) = U(i);
23             S2_Time(S2_j) = Time(i);
24             S2_j = S2_j +1;
25         case 'S3-Control'
26             S3_Budget(S3_j) = Budget(i);
27             S3_M(S3_j) = M(i);
28             S3_U(S3_j) = U(i);
29             S3_Time(S3_j) = Time(i);
30             S3_j = S3_j +1;
31         case 'S4-Control'
32             S4_Budget(S4_j) = Budget(i);
33             S4_M(S4_j) = M(i);
34             S4_U(S4_j) = U(i);
35             S4_Time(S4_j) = Time(i);
36             S4_j = S4_j +1;
37
38     end
```

```
39 end
40 subplot(4,1,1);
41
42 p1 = plot(S1_Time,S1_Budget,'--k*','LineWidth',1,'MarkerEdgeColor','k','
      MarkerFaceColor','k','MarkerSize',20);
43 hold on
44 p2 = plot(S1_Time,S1_M, '--rs','LineWidth',1,'MarkerEdgeColor','r','
      MarkerFaceColor','r','MarkerSize',10);
45 hold on
46 p3 = plot(S1_Time,S1_U,'--bs','LineWidth',1,'MarkerEdgeColor','g','
      MarkerFaceColor','g','MarkerSize',10);
47 hold off
48 xlabel('Time');
49 ylabel('Budget/M/U');
50 legend([p1 p2 p3], 'S1-Budget', 'S1-M', 'S1-U');
51
52 subplot(4,1,2);
53
54 p1 = plot(S2_Time,S2_Budget,'--k*','LineWidth',1,'MarkerEdgeColor','k','
      MarkerFaceColor','k','MarkerSize',20);
55 hold on
56 p2 = plot(S2_Time,S2_M, '--rs','LineWidth',1,'MarkerEdgeColor','r','
      MarkerFaceColor','r','MarkerSize',10);
57 hold on
58 p3 = plot(S2_Time,S2_U,'--bs','LineWidth',1,'MarkerEdgeColor','g','
      MarkerFaceColor','g','MarkerSize',10);
59 hold off
60 xlabel('Time');
61 ylabel('Budget/M/U');
62 legend([p1 p2 p3], 'S2-Budget', 'S2-M', 'S2-U');
63
64 subplot(4,1,3);
65
66 p1 = plot(S3_Time,S3_Budget,'--k*','LineWidth',1,'MarkerEdgeColor','k','
      MarkerFaceColor','k','MarkerSize',20);
67 hold on
68 p2 = plot(S3_Time,S3_M, '--rs','LineWidth',1,'MarkerEdgeColor','r','
      MarkerFaceColor','r','MarkerSize',10);
69 hold on
70 p3 = plot(S3_Time,S3_U,'--bs','LineWidth',1,'MarkerEdgeColor','g','
      MarkerFaceColor','g','MarkerSize',10);
71 hold off
72 xlabel('Time');
73 ylabel('Budget/M/U');
74 legend([p1 p2 p3], 'S3-Budget', 'S3-M', 'S3-U');
75
76 subplot(4,1,4);
77
78 p1 = plot(S4_Time,S4_Budget,'--k*','LineWidth',1,'MarkerEdgeColor','k','
      MarkerFaceColor','k','MarkerSize',20);
79 hold on
80 p2 = plot(S4_Time,S4_M, '--rs','LineWidth',1,'MarkerEdgeColor','r','
      MarkerFaceColor','r','MarkerSize',10);
81 hold on
```

```
82 p3 = plot(S4_Time,S4_U,'--bs','LineWidth',1,'MarkerEdgeColor','g','
       MarkerFaceColor','g','MarkerSize',10);
83 hold off
84 xlabel('Time');
85 ylabel('Budget/M/U');
86 legend([p1 p2 p3], 'S4-Budget', 'S4-M', 'S4-U');
```