EXACT UTILIZATION BOUNDS FOR REAL-TIME SYSTEMS
DESIGN

By

Ying Fang

B. Sc. (Electrical Engineering) Fudan University, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 1998
© Ying Fang, 1998
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical and Computer Engineering
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: \[\text{Nov. 30, 1998}\]
Abstract

Guaranteeing satisfaction of timing constraints of a real time system is an important aspect in hard real time systems design. The utilization test is an effective means to determine schedulability of a real time system prior to implementation and to assign the time budget to individual tasks in the design phase. In this thesis, a novel approach is proposed to compute the exact utilization bound for a task set with known task periods and deadlines using a fixed priority scheduling algorithm. Compared to Park’s period-specific utilization bound algorithm, this approach can obtain a more accurate utilization bound without knowledge of the exact task computation times.

In addition, this thesis presents two case studies that apply utilization bound analysis to the design phase of real-world real time systems. It also addresses some practical issues such as non-critical tasks, non-periodic tasks, and tasks without stringent periods.
Table of Contents

Abstract ii

List of Tables vi

List of Figures vii

Acknowledgments viii

1 Introduction 1

1.1 Real Time Systems ........................................ 1
1.2 Software Life Cycle for Real Time Systems .................. 2
1.3 Real Time Scheduling ....................................... 3
1.4 Motivation .................................................. 5
1.5 Related Results ........................................... 7
1.6 Contributions ............................................... 9
1.7 Outline of the Thesis .................................... 9

2 Background 11

2.1 Notations and Concepts ................................... 11
2.2 The Rate Monotonic Theory ................................ 15
2.3 The Deadline Monotonic Algorithm and the Modified Rate Monotonic Algorithm ........................................... 17
2.4 A Sufficient and Necessary Condition ..................... 18
2.5 Response Time Analysis .................................... 20
5 Conclusions and Future Work 65

5.1 Summary of Results ........................................ 65

5.2 Future Work .................................................. 67

5.2.1 Utilization Bound of a Real Time System with Shorter Dead-
      lines than Periods ........................................ 67

5.2.2 Utilization Bound of Deferred Real Time Systems ........ 68

5.2.3 Soft Real Time Systems .................................. 68

Bibliography ..................................................... 69
List of Tables

3.1 The comparison of Park’s and Liu’s algorithms. .......... 40
3.2 The comparison of Park’s and Our algorithms. .......... 41
3.3 The comparison of computation times in Park’s and our algorithms. 42
4.1 Mine control tasks and protected shared objects. ........ 55
4.2 The task set of the mine control system. ............... 57
4.3 Hydraulic motion system tasks and protected objects. .... 63
List of Figures

1.1 The life cycle of real time software development ................. 6
2.1 The diagram of the time notations in real time systems ........... 12
2.2 Time lines for two task sets ........................................... 27
2.3 The user interface of the schedulability analysis tool ............ 31
3.1 The perspective of utilization bounds ............................... 42
4.1 The system diagram of the mine pump and control system ....... 51
4.2 Design for the mine pump and control system ....................... 53
4.3 The system diagram of the hydraulic motion platform simulator system 60
4.4 Design for the hydraulic motion platform simulator system ....... 61
Acknowledgments

First and foremost, I would like to thank my supervisor, Dr. Greg W. Bond. He has introduced and guided me into this research, and provided sound advice and invaluable feedback at every stage of this project.

Without my family’s enormous support, I would not have been able to complete my study and research. In particular, I appreciate my husband Yi’s encouragement and support during my master’s program. I also would like to thank my parents and parents-in-law for taking care of my lovely girl Jane.

In addition, I would like to thank Dr. Hua Jin for being on my committee and Kendra Cooper for numerous valuable suggestions and discussions. I also wish to acknowledge Icarus Chau, Paria Noorkami, Buke Otkunc and Andrew Watkins for their help and friendship.
1.1 Real Time Systems

Real time systems, which operate in concert with real physical systems, react to externally generated input stimuli within a given deadline. A typical real time system performs one or a number of control and monitoring functions in parallel, such as data sampling, data processing and communication. Therefore, concurrent processing is a key feature of real time systems. Moreover, the response time of each event is often constrained by physical laws or conventional rules. For example, a delayed response in an air traffic control system could result in a mid-air collision of two aircraft.

Real-time systems are generally classified as hard and soft real-time systems [Burns 91]. The distinction lies in the consequences of violating the timeliness requirement: whereas soft real time environments are characterized by rising costs with increasing lateness of results, no lateness can be tolerated in hard real time environments. In other words, a late response, even if correct, is unacceptable in a hard real time system. Hard time constraints can be determined precisely by using the physical laws governing the technical processes being controlled. To satisfy timing requirements, predictability is required in hard real time systems, which means that when a task is activated, it should be possible to determine its completion time with certainty, that is, it requires the execution times and its run time schedule to be

---

²A task is a software module that can be invoked to perform a particular function. A task, like a process in Unix, is the scheduling entity in a system.
predictable. In the thesis, we will focus on discussing hard real time systems.

1.2 Software Life Cycle for Real Time Systems

As the cost of developing software has grown steadily, effective software engineering techniques and technology have drawn enormous attention since the late 1960s. One popular type of software design is a phased approach, called a software life cycle. This approach is also applied in developing real time systems. The most widely-used software life cycle is the Waterfall Model in which the following steps in time order are considered [Gomaa 93]:

1. Requirements phase: decide what the product must do and define system’s services, constraints and goals. The requirements are classified into functional requirements and non-functional requirements. The former includes system features that can be directly tested by executing the program. The latter includes specification of time constraints, programming language and the type of machine. Among them, timing requirements play a significant role in real time systems.

2. Design phase: show how the product will meet the requirements. The design process partitions the requirements to either hardware or software systems and establishes a system architecture. Furthermore, it decomposes the system into tasks.

3. Implementation phase: build the system.

4. Test phase: check if the system meets both functional and non-functional requirements.
5. Maintenance phase: maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system tasks and enhancing the system's services as new requirements are discovered.

1.3 Real Time Scheduling

In hard real time systems, failure to meet deadlines is equivalent to producing incorrect results. The sources of timing violation can be classified into three groups:

- Computation causes (corresponds to the running state): the time needed to perform task processing,
- Scheduling causes (corresponds to the ready state): the time yielded to higher priority tasks
- Synchronization cause (corresponds to the waiting state): the delay time caused by communication with the other tasks

While the computation time and synchronization time are determined by system complexity and processing load, scheduling theory defines the efficiency of exploiting the underlying system's processor. Two types of scheduling methods are widely exploited in industry and institutes.

Cyclic schedulers used to be dominant in real time design [Burns 96] and are still popular in assembly-line processing control. In this theory, after real time tasks are implemented, hand-crafted techniques are used to ensure the timing correctness by statically binding task executions to fixed slots via time lines. Under this scheduler, task periods in a real time system are harmonic and aperiodic tasks can not be handled. But, this method has turned out to be error-prone and inflexible. Moreover
timing problems can be identified only in the test phase. The iterations involved may require extensive changes to the system architecture causing cost increase and delivery delay.

Process-based scheduling algorithms, also referred as algorithmic-based scheduling theories, are developed to support task execution directly and determine which task to execute at run time based on scheduling algorithms. The advantages of process-based scheduling are as follows

- it provides analytical guarantees and gives insight into hardware and software design errors that degrade schedulability,
- it can predict run time behavior in the face of concurrency,
- it allows separation of timing and functional concerns,
- it provides an engineering basis for designing real time systems.

Process-based scheduling algorithms are usually grouped into two categories: static scheduling algorithms and dynamic algorithms. A static approach calculates schedules for tasks off-line and it requires the complete prior knowledge of tasks' characteristics. A dynamic approach determines schedules for tasks on the fly and allows tasks to be dynamically invoked. Although dynamic approaches are flexible and can adapt to changes in environment, a static scheduling technique is desirable in hard real time systems. This is because no event should be unpredicted and schedulability should be guaranteed before execution for a hard real time system. Since our concern in the thesis is hard real time systems, we will restrict ourselves to only discussing of static scheduling techniques.

The main goal of real time task scheduling is to provide a schedule that allows all tasks to be processed in time according to tasks' deadlines, so the scheduling algorithm
must map tasks onto resources such that all tasks meet their time requirements. Therefore, it must be possible to show that a scheduling algorithm applied to real time systems fulfills the timing requirements of tasks.

1.4 Motivation

Attempts have been made to address real time scheduling problems and to test the timing requirements since 1970s. The process-based scheduling algorithms have been accommodated in the conventional life cycle of software development. By doing so, the life cycle for real time software development is extended to separately analyze timing requirements to construct a system model and to estimate the utilization threshold of the processor as shown in Figure 1.1.

The benefit of performing analysis in the early development life-cycle is identifying potential performance problems in the design phase before they become implementation errors because fixing errors early is less expensive than fixing them later on. This thesis focuses on studying timing requirements involved in real time systems design during the “system model and analysis” phase. Currently, a simulation-based approach is the only approach applied at the analysis stage. This technique first uses a domain expert to estimate task execution times. Then a simulator (e.g. [Smith 93, Woodside 95]) uses this information to analyze system performance and identify potential bottlenecks. The validity of the simulation results are only as valid as the estimated task execution times. Furthermore, the simulation results provide no information regarding the overall utilization requirements of processing elements. In a related approach [MajumdarEtA 92, Majumdar 92], the bounds on task response times are stochastically characterized and computed given estimations of their computation time characteristics. The computed response times can then be compared
Figure 1.1: The life cycle of real time software development
with the required deadlines in order to identify requirements violations.

This approach uses an inefficient "generate and test" method to real-time systems design. First, task execution times are generated by domain experts, and then the ability of a system to meet its deadlines is tested. If the test fails (i.e. if the system misses one or more deadlines), then new task execution times are generated and the system is re-tested. This process stops when a set of task execution times is identified that ensures system deadlines are met. Alternately, when no realistic task deadlines can be identified, the system is re-designed, at which point the process begins again.

In contrast, this thesis investigates a novel approach to design analysis called utilization bound analysis. Compared to simulation-based analysis, utilization bound analysis is more relevant for the design or maintenance of a real-time system since it is important to know as early as possible the permissible execution time bounds of tasks. With this information, designers and maintainers are able to assess for a given design whether the predicted execution time bounds are realistic for the tasks with unknown execution times or whether a modified design is required. This information can also be used as execution time guidelines for implementors of system components during design or maintenance.

1.5 Related Results

When a real time system is scheduled by a static scheduling algorithm, the processor utilization is upper bounded. This bound depends on task parameters such as the number of tasks, task execution times, task periods, and task deadlines.

A utilization test was first used by Liu and Layland [Liu 73] to examine whether hard real time periodic tasks are schedulable using the rate monotonic scheduling algorithm. They demonstrated that all tasks are schedulable when the utilization of
processor is less than the worst case utilization bound. However, their computational model is very restrictive: the deadline of a task must be the same as its period; and the utilization threshold is pessimistic: many task sets are schedulable with much higher utilization than the estimated utilization bound from Liu and Layland's theory.

Following Liu and Layland's pioneering work, a fair amount of literature has been dedicated to extending the original system utilization bound theory.

- Leung and Whitehead introduced a new fixed priority scheduling algorithm to deal with a real time system whose task deadlines are less than the corresponding periods.

- Lehoczky and Sha [Lehoczky 86], and Peng and Shin [Peng 89] generated an approach to calculate the utilization bound for the case in which task deadlines are a constant factor $\Delta$ of the corresponding task periods in a real time system.

- Park [Park 93] developed a technique to determine the utilization bound by using linear programming for fixed priority scheduling algorithm and arbitrary task deadlines.

Park's work extended the utilization test to any static scheduling theory and obtained more accurate utilization bound than Liu's work did. But his approach is not capable of finding the exact utilization bound for some task sets with known periods and deadlines. Furthermore, although Park suggests that utilization bound analysis can be used for design analysis, the utility of this approach has not yet been investigated.
Chapter 1. Introduction

1.6 Contributions

As part of the evaluation, we uncover a flaw in one of Park’s corollaries [Park 96]. We show that this flaw causes his procedure to calculate incorrect utilization bound in some instances. We then present a corrected version of the corollary.

We describe a graphical schedulability analysis tool that was implemented to investigate the run-time system behaviors. This tool can verify the schedulability of a given hard deadline task set for a given fixed priority scheduling algorithm.

Furthermore, we propose an extension to Park’s procedure that has following advantages:

1. The calculated utilization bound is accurate.
2. Task deadline can be less than its period.
3. It is applicable to all fixed-priority real time systems.

Finally, we apply the extended approach to two realistic real time systems designs to demonstrate the capability of this approach. We also analyze the common limitation in utilization bound estimation methodology by using linear programming for a real time system with very short deadlines. A suggestion is also given to work around this limitation.

1.7 Outline of the Thesis

Following the introduction of the thesis topic in the first chapter, Chapter 2 reviews the related previous work on real time scheduling and schedulability tests based on fixed priority assignments. It also introduces a schedulability analysis tool developed during thesis study to visualize the task run time behaviors.
Chapter 3 presents the exact utilization bound procedure to evaluate exact utilization bounds based on the parameters of a task set available at the design phase of development.

Chapter 4 demonstrates the application of the exact utilization bound procedure to two case studies. In addition, it also discusses some practical problems raised during the process of design and timing requirement validation.

We summarize our work and recommend future work in Chapter 5.
Chapter 2

Background

In this chapter, we provide an overview of the widely used approaches for scheduling hard real-time systems and for testing the schedulability of a real-time system. We also describe how one published approach is flawed and describe how the flaw can be corrected. In addition, a graphic schedulability tool is outlined which facilitates not only the thesis research, but also real-time systems design by graphically depicting run-time behaviors.

2.1 Notations and Concepts

A real-time system with \( n \) tasks is represented as a task set \( \{ r_1, r_2, ..., r_n \} \) in which the tasks are sorted in the decreasing order of the priorities to be scheduled by fixed priority assignment. In this thesis, we will use the convention that \( r_1 \) has the highest priority and \( r_n \) has the lowest priority. The timing constraints of each task \( r_i \) are specified in terms of the following parameters, shown in Figure 2.1:

- \( C_i \) is the computation (or execution) time of task \( r_i \)—in hard real-time systems, it requests the worst case computation time,
- \( T_i \) is the period time of task \( r_i \),
- \( D_i \) is the deadline of task \( r_i \),
- \( I_i \) is the phase of task \( r_i \), referred to initial release time relative to some common
Chapter 2. Background

Figure 2.1: The diagram of the time notations in real time systems

- $A_i$ is the arrival time or release time of the task. The $k$th arrival time of periodic task $\tau_i$ is defined as $kT_i + I_i$ and the event is called as $k$th request of task $\tau_i$.

- $R_i$ is the response time which is between the arrival time and the complete time of task $\tau_i$.

Each parameter forms its own set such as periodic set $T$, deadline set $D$, computation set $C$, and etc.

In real time systems, the ability to meet deadlines is paramount. We describe a task as schedulable when it meets its deadline or its maximum response time is less than the deadline. A real time system is schedulable when all tasks in the system are schedulable. The terms “feasible” and “schedulable” are interchangeably used in the thesis.

Special notations for the exact feasible utilization bound and period-specific utilization bound techniques are further defined in the following.
When we discuss a task set, the following corresponding parameter sets are assumed to be known:

- Period time set: $T$,
- Deadline time set: $D$,
- The phase of each task is set to zero.

Since computation time is not required for utilization evaluation, the computation time set is treated as an unknown. In the remainder of this thesis, a task set will be understood to represent the task set itself along with the sets of known parameters. If, however, computation time constraints are known for a task, then these can be included as known parameters. Therefore, any computation time can be either a constant value or a range. Initially the range is set as $[0,T_i]$.

The priority $i$ subset of a task set is defined as the subset of the task set containing the first $i$ elements where $i$ is less than $n$. Therefore the priority 1 subset of \{$r_1, r_2, ..., r_n$\} is \{$r_1$\}, the priority 2 \{$r_1, r_2$\}, and the priority $i$ is \{$r_1, r_2, ..., r_i$\}. The total number of the subsets are $n$. A system consisting of the priority $i$ subset is also called as priority $i$ subsystem.

The processor utilization is defined as the fraction of processor time spent in the execution of the task set as

$$U = \frac{T_e}{T_e + T_i},$$

where $T_e$ is the execution time, and $T_i$ is the idle time. The task utilization is referred to the fraction of computation time during the period as

$$U_{\text{task-name}} = \frac{C}{T}.$$
So, the processor utilization $U_n$, processing $n$ tasks, is:

$$U_n = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$

From another point of view, $U_n$ is also the system utilization for a task set with $n$ tasks.

Informally, the utilization bound in a real time system is the maximum utilization with the requirement that all tasks satisfy their deadlines at any instant. And the exact utilization bound means that a real time system is just feasible with a special computation time set. When the system utilization is above the exact utilization bound, there exists a possible set of computation times for which the task set is not feasible. Just feasible is defined in [Park 93]:

A particular task set with some given periods $T_i$, deadlines $D_i$, computation times $C_i$ and priority assignments $P_i$ is said to be just feasible if it meets the following conditions:

1. the task set meets all its deadlines in its present form,
2. $\exists j$ that if the computation time $C_j$ of task $\tau_j$ were changed to $C_j + \epsilon$, where positive $\epsilon \to 0$, then at least one of the tasks in the task set would miss at least one of its deadlines.

In summary, there are two key factors in the utilization bound model: feasibility (or schedulability) and full utilization (no idle time during a time interval).

When the utilization of the task set with any computation set is below this bound, a fixed priority assignment will make the system feasible.

In the remainder of this chapter, we will provide an overview of the following definitions of utilization bounds to distinguish the different approaches:
Chapter 2. Background

- B(E): the exact feasible utilization bound.
- $B_i$: the utilization bound of the priority i subsystem.
- $U_i^*$: the ith individual utilization bound calculated by Park's algorithm.
- B(Liu): Liu and Layland's bound for the rate monotonic algorithm.

2.2 The Rate Monotonic Theory

The rate monotonic scheduling algorithm is a preemptive priority-driven algorithm developed by Liu and Layland [Liu 73]. In this model, a system receives periodic events from the external environment and these events are not buffered. Each event initiates a task to handle it. The priority of each task is statically assigned according to the task period where tasks with shorter periods have higher priorities. The following assumptions are also made about the environment:

1. The application consists of a fixed set of tasks.
2. All tasks are periodic.
3. All tasks are independent.
4. All system overheads, context switching times and so on are ignored.
5. All tasks have a deadline equal to their periods.
6. All tasks have a fixed worst-case execution time.

Although these assumptions are very stringent, the theory focuses on the characterizations of a task by two parameters: its request period and its computation time. Based on the above assumptions, Liu and Layland derived three important results:
Chapter 2. Background

Theorem 2.1 (Liu and Layland) A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.

The longest response time for any task of \( r_t \) for the first request of of \( r_t \) occurs when all tasks arrive at a critical instant which means \( I_1 = I_2 = \ldots, = I_n = 0 \). So a realtime system is schedulable if all tasks arriving at their critical instants meet thesis respective first deadlines.

Theorem 2.2 (Liu and Layland) If a feasible priority assignment exists for some task set, the rate-monotonic priority assignment is feasible for that task set.

This means that the rate monotonic scheduling scheme is an optimal algorithm.

Theorem 2.3 (Liu and Layland) A periodic task set \( \tau_1, \tau_2, \ldots, \tau_n \) with \( D_i = T_i, \ 1 \leq i \leq n \), is schedulable by the rate monotonic scheduling algorithm if

\[
U_n = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq B(Liu) = n(2^{1/n} - 1). \tag{2.1}
\]

The three theorems provide a theoretical basis to study and test system schedulability. However, the threshold utilization is pessimistic. In the uniprocessor case, the bound is \( \log e2 = 0.693 \), meaning that if periodic utilization is kept below this level, the optimal fixed priority algorithm will meet all deadlines. But many task sets are schedulable with much higher utilization bounds. In fact, Liu and Layland provided an exact utilization bound formula.

If a task set whose ratios between any two task periods are less than 2, the exact upper utilization bound is

\[
B(Liu) = 1 + g_1\left[\frac{g_1 - 1}{g_1 + 1}\right] + \sum_{i=2}^{n} g_i\left[\frac{g_i - g_{i-1}}{g_i + 1}\right] \tag{2.2}
\]

where \( g_i = (T_n - T_i)/T_i, \ i = 1, 2, \ldots, n. \)
Then if a task set with one or more ratios between the period and the longest period are greater than 2, a corresponding task set with maximum period ratios less than 2 can be created where the new utilization bound is less than the previous one. Suppose for task $\tau_i$, \([T_n/T_i] > 2\). Let $T_n = qT_i + r$, $q > 1$ and $r \geq 0$. The corresponding task $\tau'_i$ replaces the task $\tau_i$, such that $T'_i = qT_i$ and $c'_i = c_i$. In order to maintain the full utilization feature, the computation time of the lowest priority task, $c_n$, is increased at most $c_i(q - 1)$. The utilization of the corresponding task set is denoted as $U_n'$.

\[
U_n' - U_n = c_i(q - 1)[\frac{1}{qT_i + r} - \frac{1}{qT_i}]
\]  

(2.3)

Since $q - 1 > 0$ and $[1/qT_i + r - 1/qT_i] \leq 0$, then $U_n' \leq U_n$. Therefore the system utilization of $U_n$ is equal to or greater than that in its corresponding task set which has the exact utilization bound. As a result the exact utilization bound of the corresponding task set is regarded as the utilization bound of this task set.

Furthermore, $n(2^{1/n} - 1)$ is the lowest value among all exact utilization bounds for any task sets with $n$ tasks.

Therefore, the gap between the utilization bound given in Equation 2.1 and the exact value is due to

1. the transformation of a task set to meet the requirement that ratios of any periods is less than 2,

2. the minimization of the exact utilization bound.

### 2.3 The Deadline Monotonic Algorithm and the Modified Rate Monotonic Algorithm

Since the rate monotonic algorithm is optimal under the condition that the task deadline must be the same as its period and the schedulability test formula is only
suitable for a system using rate monotonic priority ordering, a more exact criterion is required for general circumstances. Leung and Whitehead introduced the deadline monotonic algorithm for real time systems in which task deadline is less than its period [Leung 82] and Shih et al. showed a modified rate-monotonic algorithm when tasks have deferred deadline [Shih 93].

The deadline monotonic algorithm is another fixed priority scheduling algorithm. It assigns the task priority inversely with respect to task deadlines, that is task $\tau_i$ has a higher priority than task $\tau_j$ if $D_i < D_j$. Leung and Whitehead proved the optimality of the deadline monotonic algorithm in [Leung 82]

**Theorem 2.4 (Leung and Whitehead)** For a periodic task set $\tau_1, ..., \tau_n$ with $D_i \leq T_i, 1 \leq i \leq n$, the optimal fixed priority scheduling algorithm is the deadline monotonic scheduling algorithm. A task set is schedulable by this algorithm if the first task of each task after a critical instant meets its deadline.

Since there is no effective schedulability test for the deadline monotonic algorithm, each task must be checked if it meets its deadline in the worst case after implementation to determine the schedulability of a task set.

On the other hand, the modified rate monotonic algorithm is a semi-fixed priority-driven algorithm in which each task is assigned two fixed priorities, the higher priority for its old requests and the lower priority for the current request. Since the performance of the modified rate monotonic algorithm becomes worse when the period ratio increases, we will not cover it in the thesis.

### 2.4 A Sufficient and Necessary Condition

An exact analysis of the schedulability of a task set with a known computation time set using a fixed priority scheduling algorithm is proposed by Lehoczky et al
This technique examines a specific set of schedules at a finite number of times, one schedule for each task. If task \( \tau_i \) completes execution at one of these specific times using the schedule constructed for that task, then all fixed-priority preemptive schedules for that workload are feasible. This result is exact when task arrivals are assumed to be independent and might occur simultaneously, that is, no assumptions are made about the initial arrival times of the tasks, and this result is said to be exact for all task phasings.

The schedule used to analyze the feasibility of a task \( \tau_i \) is defined by the workload \( W_i(t) \) at a critical instant. The workload due to all tasks with higher priority than \( \tau_i \) is referred as a function of time

\[ W_i(t) = \sum_{j \neq i}^m C_j \left\lfloor \frac{t}{T_j} \right\rfloor. \]  

The times at which this schedule must be checked are the scheduling points (arrival times) for tasks \( \tau_1 \ldots \tau_{i-1} \) up to \( \tau_i \). If \( \tau_i \) completes by any of these scheduling points, then \( \tau_i \) will always be feasibly scheduled. Theorem 2.4 gives a necessary and sufficient condition for the first request of each task to meet its deadline.

**Theorem 2.5 (Lehoczky)** Given periodic tasks \( \tau_1, \ldots, \tau_n \),

1. Task \( \tau_i \) can be scheduled for all task phases using the rate monotonic algorithm if and only if

   \[ L_i = \min_{0 \leq t \leq T_i} \frac{W_i(t)}{t} \leq 1 \]  

2. The entire task set can be scheduled for all task phases using the rate monotonic algorithm if and only if

   \[ \max_{0 \leq t \leq n} L_i \leq 1 \]  

The characterization of schedulability given in Theorem 2.4 requires a minimization over the continuous variables \( t \in [0, T_i] \). In fact, only a finite number of times \( t \)
need to be checked. The function \( W_i(t)/t \) is strictly decreasing except at a finite set of values called the rate monotonic scheduling points where the function has a local minimum. The condition can be rewritten as:

\[
L_i = \min_{t \in s_i} \frac{W_i(t)}{t} \leq 1
\]  

where \( s \in S_i = \{kT_u | u = 1..i, k = 1..\lceil T_i/T_u \rceil \} \).

2.5 Response Time Analysis

Another sufficient and necessary condition of schedulability test is the worst-case response time. The worst-case response time analysis can be applied to verify the schedulability thoroughly. The response time is the duration between a task's arrival time and its completion. When all response times of tasks in a real-time system are less than or equal to their deadlines, the system is feasible.

The model of the worst-case response time analysis is more general than the rate monotonic model by relaxing the restriction that the deadline is equal to the period. Moreover, the priority assignment can be any static one. Assume a real time system with a set of tasks \( \{\tau_1, \tau_2, ..., \tau_n\} \).

The fact that a task with higher priority preempts the execution of those with lower priority leads to the conclusion that for all priorities \( i > j \), the response time of task \( i \) is greater than that of task \( j \) and the response time of \( i \) is affected by all the tasks with higher priority when all tasks release at the critical instant. This analysis requires calculating the worst-case response time of each task by the following equation [Joseph 86]:

\[
RT_i = \sum_{j=1}^{i-1} \left[ \frac{RT_j}{T_j} \right] \times C_j + C_i
\]  

where the \( RT_i \) is the worst-case response time. In the equation, the first part is the sum of the computational requirements for all tasks with higher priority occurring in
the interval $[0, RT_i)$, and the second part is the computational requirement for itself. Unfortunately, the response time is hard to solve from Equation 2.8.

Joseph and Pandya [Joseph 86] introduced an approach to calculate the response times for a task set using the rate monotonic scheduling algorithm. The approach is based on the recognition that once the execution of task $i$ is preempted, the instance of its resumption is independent of the order of arrival of requests at a higher priority level. First we define a time interval $[t, t')$ which contains all values from $t$ up to $t'$, but not including $t'$. During this interval the maximum number of requests from task $j$ is

$$\text{Inputs}([t, t'), j) = \left\lceil \frac{t'}{T_j} \right\rceil - \left\lfloor \frac{t}{T_j} \right\rfloor$$

(2.9)

and its required computation time is

$$\text{Comp}([t, t'), j) = \text{Inputs}([t, t'), j) \times C_j$$

(2.10)

So the computation time needed for all invocations at levels greater than $i$ can be defined

$$\text{Comp}([t, t')) = \sum_{j=1}^{i-1} \text{Inputs}([t, t'), j) \times C_j$$

(2.11)

For calculation of the response time of task $i$, we first initialize $t'=t_0+Ci$. If $\text{Comp}([t_0, t'])$ is zero, no preemption occurs. Task $i$ uses the whole time from $t$ to $t'$. Then the response time is $t' - t$. Otherwise we extend $t'$ to $t' + \text{Comp}([t_0+t'])$ which means extra time is needed to handle the interference of preemption. We substitute $t$ with $t'$ and $t'$ with $t' + \text{Comp}([t_0+t'])$ and calculate the $\text{Comp}([t, t'])$ until we find $\text{Comp}([t, t'])$ or $t'$ exceeds the deadline of task $i$ which means the task $i$ couldn't finish its execution before its deadline.

The above procedure can be generalized to the following recursive definition of function $R(t, t', i)$.
Response(t,t',i) =
if Comp([t,t'])=0 then t'
else Response(t,t'+Comp([t,t']),i)

The worst-case response time at level \( i \) can be defined as

\[ R_i = \text{Response}(0,C_i,i) \]  \hspace{1cm} (2.12)

where the task \( i \) releases at the critical instant.

### 2.6 Park’s Algorithm

Park et al. published two articles [Park 95, Park 96] on period-specific utilization bound algorithm. The first one focuses on the detailed procedure of calculating utilization bound and the second one talks about the proofs of the algorithm. First, the overview is presented. Then, we discuss a flaw in the second paper.

#### 2.6.1 Period-Specific Utilization Bound I

The period-specific utilization bound I procedure (PSUB-I) [Park 95] provides a more accurate utilization bound than Liu and Layland’s theory by accommodating task periods and deadlines. It also extends its usage range to any fixed-priority scheduling algorithm. The following assumptions are made in this model:

1. All tasks are periodic.
2. All tasks are independent.
3. All system overheads are ignored.
4. All tasks are assigned a fixed unique priority, and scheduled by priority-based preemptive algorithm.
The objective of this technique is to determine the exact utilization bound without the knowledge of task computation times. In order to determine the system utilization bound $B(Park)$, a series of individual utilization bounds $U_1^*, U_2^*, ..., U_n^*$ are determined such that individual task $\tau_i$ will meet all its deadlines whenever the combined utilization of tasks $\tau_1, \tau_2, ..., \tau_i$ is less than $U_i^*$. When the task deadlines are deferred, $U_{ij}^*$ is defined as the utilization bound of the $j$th request of task $\tau_i$ which means the $j$th request of task $\tau_i$ meets its deadline whenever the utilization of $\tau_1, \tau_2, ..., \tau_i$ is below $U_{ij}^*$. So, the individual utilization bound depends on $U_{ij}^*$. Linear programming is used to evaluate $B_{ij}$ by formulating a set of constraints with the deadline requirement for the scheduling algorithm.

The procedure to evaluate $U_{ij}^*$ is

\[
\text{minimize} \quad U_{ij}^* = \sum_{k=1}^{i} \frac{C_k}{T_k} \tag{2.13}
\]

subject to

\[
\forall_{m=1}^{i}, \quad \forall_{l=1}^{\lfloor \frac{D_{ij}}{T_l} \rfloor} \sum_{k=1}^{i} \left[ \frac{t_m}{T_k} C_k + \min\left(j, \frac{t_m}{T_l} C_l\right) \right] \geq t_m \tag{2.14}
\]

and

\[
\sum_{k=1}^{i} \left[ \frac{D_{ij}}{T_k} C_k \right] \geq D_{ij} \tag{2.15}
\]

and

\[
\forall_{k=1}^{i} \quad C_j \geq 0 \quad 1 \leq j \leq i. \tag{2.16}
\]

Then the individual utilization bound is

\[
U_i^* = \min(U_{ij}^*) \tag{2.17}
\]
and the utilization bound is

\[ B(Park) = \min(U_i^*) \]  \hspace{1cm} (2.18)

Equation 2.14 ensures no idle time is permitted prior to the release point of each task in the interval \([0, lT_m]\). Minimization in this inequality ensures that for task \(\tau_i\), the executions of requests later than the \(j\)th request are not taken into consideration in the process of determining that there is no idle time because considering up to request \(j\) is required as the deadline is greater than the corresponding period. The inequality in Equation 2.15 states that there should be no idle time during deadline \(D_{ij}\). The iterative calculation \(U_i^*\) will terminate when \(j\) reaches \([\text{LCM}(T_1, T_2, ..., T_n)/T_i]\), where LCM represents the least common multiple, or there is no solution for the \(j\)th request. This means \(\tau_i\) has finished by the time \(jT_i\) for any set of computation times so there will be no extra computation demand subsequent to the request of \(\tau_i\).

In some circumstances, this approach calculates the exact utilization bound. This will be discussed further in chapter 3.

### 2.6.2 Period-Specific Utilization Bound II

Park further proposed the period specific utilization bound II (PSUB-II) procedure in which he focused on a model that task deadlines are not greater than their periods [Park 96]. He also gave some useful conditions and theorems.

**Condition 2.1** (Park). There is no idle processor time prior to the first deadline of task \(\tau_i\).

**Condition 2.2** (Park). All tasks with higher priority that arrive before the first deadline of task \(\tau_i\) must be finished before the first deadline of task \(\tau_i\).
Corollary 2.1 (Park). \( \sum_{j=1}^{i} C_j \times \left\lfloor \frac{D_i}{T_j} \right\rfloor = D_i \) if and only if the previous two conditions are true.

Theorem 2.6 (Park). The utilization bound \( U_i^* \) for the schedulable task \( r_i \) can be achieved by the following procedure: Minimize

\[
U_i^* = \sum_{j=1}^{i} \frac{C_j}{T_j}
\]

subject to

\[
\sum_{j=1}^{i} C_j \times \left\lfloor \frac{D_i}{T_j} \right\rfloor = D_i
\]

and

\[
C_j \geq 0 \quad 1 \leq j \leq i.
\]

The condition \( \sum_{j=1}^{i} C_j \times \left\lfloor \frac{D_i}{T_j} \right\rfloor = D_i \) is intended to constrain the task set to be just feasible. The following theorem follows the Corollary 2.1.

Theorem 2.7 (Park). The period specific utilization bound (PSUB) for the entire system is the smallest \( U_i^* \) among all tasks. That is,

\[
PSUB = \min_{m=1}^{n} U_m^*
\]

2.6.3 Discussion

Park's PSUB-II procedure, as published, has a problem in Corollary 2.1 which we illustrate here by two examples. The first example shows that the solutions obtained using Park's procedure permit idle time during the first deadlines of some tasks, which means that the solution is sub-optimal. The second example shows that the procedure does not guarantee the schedulability of the task set.

According to the definition of a just feasible task set, there should be no idle time in the interval \([0,D_n]\). Otherwise, there is some idle time \( \Delta \) before the first deadline.
Then, the computation time of task \( \tau_n \) can be increased up to \( \Delta \) and the task set is still feasible. But Park’s procedure does not lead to correct results in the following two instances:

**Instance 1:** a task set \( \{\tau_1, \tau_2, \tau_3\} \)

\[ \tau_1 = \{C1, 10, 10, 0\}, \quad \tau_2 = \{C2, 12, 12, 1\}, \quad \tau_3 = \{C3, 25, 25, 2\} \]

When \( C = \{C_1, C_2, C_3\} = \{2.5, 5, 2.5\} \), the utilization bound is 0.7583.

**Instance 2:** a task set \( \{\tau_1, \tau_2, \tau_3\} \)

\[ \tau_1 = \{C1, 7, 7, 0\}, \quad \tau_2 = \{C2, 12, 12, 1\}, \quad \tau_3 = \{C3, 47, 47, 2\} \]

When \( C = \{C_1, C_2, C_3\} = \{5, 3, 0\} \), the utilization bound is 0.9640.

Here each task is represented by four parameters in order: computation time, period, deadline, and priority.

Both examples satisfy the condition \( [T3/T1] \times C1 + [T3/T1] \times C2 + C3 = 25 = D_3 \). Figure 2.2 demonstrates the run time behavior of the two systems. In A, task 2 delays its completion for 4.5 units which is equal to the sum of total idle time so the condition does not recognize the existence of idle time. The deferred completion violates Condition 2.1 as well. In B, besides the ignored idle time, task 2 misses its first deadline which can not be guaranteed by the constraints.

As a solution to the idle time problem, Corollary 2.1 should be modified as

**Corollary 2.2** \( \sum_{j=1}^{i} C_j \times [D_i/T_j] = D_i \) if the two previous conditions are true.

Proof. If there is no idle time prior to the first deadline of task \( \tau_i \), there exists one task running at any instance during \([0, D_i)\). Since the number of each task \( j < i \) released in the period of \( D_i \) is \([D_i/T_j]\), the total execution time of task \( \tau_j \) is \( C_j \times [D_i/T_j] \). Therefore the total execution time at priority levels equal to or less than \( i \) is \( \sum_{j=1}^{i} C_j \times [D_i/T_j] \) should be greater than or equal to \( D_i \). In Condition 2,
Figure 2.2: Time lines for two task sets
all tasks with higher priority that arrive before the first deadline of task $\tau_i$ must be finished before the first deadline of task $\tau_i$. Then the total execution times of all tasks with higher priority should be less than or equal to $D_i$. Moreover task $\tau_i$ must finish its own computation before $D_i$, otherwise its deadline is missed. So $\sum_{j=1}^{i} C_j \times \lfloor D_i/T_j \rfloor$ should be less than or equal to $D_i$.

Therefore, $\sum_{j=1}^{i} C_j \times \lfloor D_i/T_j \rfloor = D_i$. QED

However, if $\sum_{j=1}^{i} C_j \times \lfloor D_i/T_j \rfloor = D_i$, it’s possible that there might exist idle time prior to the first deadline of task $\tau_i$ as illustrated by the aforementioned examples. The condition that there is no idle time at each task release points is therefore required to constrain the solution.

In both of Park’s procedures to calculate $U_i^*$, feasibility constraints are excluded. Therefore the feasibilities of all tasks but $\tau_i$ in a task subset $\{\tau_1, \tau_2, ..., \tau_i\}$ won’t be guaranteed. Due to the nature of the linear programming, the inequalities ensure that one of the deadlines will just be met. That means that the linear programming will obtain the optimum value when it reaches one of the inequalities. In the case of $D_i \leq T_i$, each individual utilization bound $U_i^*$ only guarantees the deadline of task $\tau_i$ is met. Since only one task is guaranteed to meet its deadline, the procedures are iteratively performed for each task to ensure that all deadlines will be met.

### 2.7 Schedulability Analysis Tool

In contrast to the aforementioned mathematical treatments of schedulability tests, the graphic method for schedulability test is not only sufficient and necessary, but it also gives detailed indication of the actual response times of tasks and the computation time left after a task misses its deadline. The schedulability tests only give a
"yes" or "no" answer to a tested system. A graphic schedulability analysis tool also can illustrate the run time behavior of a process according to the scheduling algorithm. Gantt charts and time-lines, such as those represented in Figure 2.2, are two useful ways of demonstrating execution patterns. Schedulability analysis proceeds by starting the execution in the critical instant represented at time 0.

The schedulability analysis tool (SAT) developed for the thesis is an extended graphic test tool to automatically create a Gantt chart for a given real time system under a given scheduling algorithm to test the schedulability of the system, illustrated in Figure 2.3. SAT provides an X windows-based colored graphical user interface environment. The graphical user interface consists of three types of windows: the display window, the task information window and the control panel window. In the display window, Gantt charts illustrate the execution slots of each task over time. Different tasks are denoted by different colors. The heights of rectangles of task execution are classified into 3 types where the highest rectangle represents the execution of the first request; the shortest one is for consecutive requests; and the rectangle with medium height denotes the idle time in the processor. Only the first request is illustrated in the graph because this is sufficient to test the schedulability of a real time system with the rate monotonic or deadline monotonic algorithms. The time interval is up to the least common multiple of all task periods which is required for a real time system with deferred deadlines.

The task information windows present task parameters such as task name, response time etc. The control panel window provides the user with buttons for inputting task parameters, assigning task priorities satisfying the rate monotonic algorithm, plotting the Gantt chart and performing other related functions.

The schedulability analysis tool is written in TCL/Tk because of its flexibility and productivity. Since TCL/Tk is an interpreted language, it's convenient to design
a prototype and fix bugs. The low execution efficiency can be circumvented by re-implementing parts of scripts in C, because TCL/TK provides the interfaces to other languages such as C. Moreover, with the help of Tk, a toolkit for the X windows system, building user interface becomes a pleasurable job.
Figure 2.3: The user interface of the schedulability analysis tool
Chapter 3

Exact Feasible Utilization Bound

The estimated system utilization bound based on Park's algorithm departs from the actual value because of the minimization of the individual utilization bounds. The minimization step must be performed since the constraints in the Park's procedure to obtain the individual utilization bound do not guarantee the feasibility of tasks other than the one with the lowest priority. Our proposed algorithm of exact feasible utilization bound takes into account the feasibility constraints of other tasks. In our model, the task deadlines do not exceed their corresponding periods.

3.1 Overview of the Algorithm

Our algorithm is based on linear programming to compute the exact utilization bound with three sets of constraints for a real time system:

1. Full utilization constraints which guarantee no idle time is allowed during the assigned time interval.

2. Task schedulability constraints which guarantee all tasks meet their corresponding deadlines.

3. Computation time constraints which limit the computation time ranges. We set the computation time of each task to be not less than 0 and not greater than its deadline.
Then minimization is performed on the system utilization to determine the lowest utilization for a specific task set. Our algorithm accommodates task schedulability constraints which do not exist in Park’s algorithm. Both full utilization and task schedulability constraints guarantee the feasibility of a real time system whose utilization is under the minimal utilization.

To find a set of task schedulability constraints we adopt the iterative method to obtain system utilization bounds of its priority subsets. Each subsystem utilization bound $B_k$ of a subset $\{\tau_1, ..., \tau_k\}$, $k \geq 1$ is regarded as a constraint for its superset $\{\tau_1, ..., \tau_k, \tau_{k+1}\}$ in order to calculate the bound $B_{k+1}$. Since $B_1, ..., B_k$ guarantee the feasibilities of tasks $\tau_1, ..., \tau_k$ and the full utilization constraints can guarantee the feasibilities of task $\tau_{k+1}$, then we obtain a feasible subsystem utilization bound. Hence, the system utilization bound is the least priority task set utilization bound.

This method is demonstrated below. Assume there are three tasks $\{\tau_1, \tau_2, \tau_3\}$ in a system along with a periodic time set and a deadline set where the ratio of any two periods is less than 2. Since $0 < C_1 < D_1 < T_1$, then $B_1 = 1$. $B_2$ is processed with full utilization constraints:

\[
C_1 + C_2 \geq D_1
\]

\[
\left[ \frac{D_2}{T_1} \right] C_1 + C_2 \geq D_2
\]

and task schedulability constraints:

\[
\frac{C_1}{T_1} \leq B_1
\]

and computation time constraints:

\[
0 \leq C_1 \leq D_1 \leq T_1
\]

\[
0 \leq C_2 \leq D_2 \leq T_2.
\]
Then $B_3$ is determined by full utilization constraints:

$$C_1 + C_2 + C_3 \geq D_1$$
$$\left\lfloor \frac{D_2}{T_1} \right\rfloor C_1 + C_2 + C_3 \geq D_2$$
$$\left\lfloor \frac{D_3}{T_1} \right\rfloor C_1 + \left\lfloor \frac{D_3}{T_2} \right\rfloor C_2 + C_3 \geq D_2$$

and task schedulability constraints:

$$\frac{C_1}{T_1} \leq B_1$$
$$\frac{C_1}{T_1} + \frac{C_2}{T_2} \leq B_2$$

and computation time constraints:

$$0 \leq C_1 \leq D_1 \leq T_1$$
$$0 \leq C_2 \leq D_2 \leq T_2$$
$$0 \leq C_3 \leq D_3 \leq T_3.$$ 

Consequently, we obtain the system utilization $B$ equal to $B_3$ as $U_1 \leq B_1$, and $U_2 \leq B_2$.

### 3.2 Theoretical Foundations of the Algorithm

**Lemma 3.1**: When the subsystem utilization bound $B_i$ is obtained only subject to the full utilization constraints:

$$\forall_{m=1}^{i} \sum_{j=1}^{m} \left\lfloor \frac{D_m}{T_j} \right\rfloor C_j \geq D_m$$

the corresponding deadline $D_i$ is always met.
Proof. Assume $\sum_{j=1}^{i}[D_i/T_j]C_j > D_i$. According to lemma 2.2, tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ complete their execution before $D_i$, so task $\tau_i$ doesn't finish its execution until $D_i + \delta$. We also know there is no idle time before $D_i$, otherwise task $\tau_i$ would execute during the idle time. We create a different set of computation times so that

$$C'_i = C_i - \delta$$

We also know there is no idle time before $D_i$, otherwise task $\tau_i$ would execute during the idle time. In this case, the new computation set still maintains full utilization. This transformation, as well, preserves the schedulability of the task set because task $\tau_i$ completes before its deadline $D_i$. Hence, the new utilization bound is

$$B'_i = B_i - \frac{\delta}{T_i}.$$ 

Since $\delta/T_i$ is always positive, the new utilization can only be smaller than or equal to the original one. This violates the assumption that $B_i$ is minimal.

Therefore, with the full utilization constraints, the schedulability of task $\tau_i$ is guaranteed.

**Lemma 3.2** For a task set $\{\tau_1, \ldots, \tau_n|n \geq 2\}$, $B_i$ is the utilization bound of its priority $i$ subset. In fixed-priority preemptive real time systems, tasks $\tau_1, \tau_2, \ldots, \tau_n$ are still schedulable if and only if

$$\forall_{m=1}^{n} \sum_{i=1}^{m} \frac{C_i}{T_i} \leq B_m.$$ 

Proof. First the sufficiency is proved by induction on number of tasks. The system utilization of one task is 1. When $C_1/T_1 \leq 1$, task $\tau_1$ is schedulable. For a 2-task system, task $\tau_2$ is also schedulable as $C_1/T_1 + C_2/T_2 \leq B_2$ by Lemma 3.1.
On the other hand, when task $\tau_1$ is schedulable, the computation time must be less than its deadline. Therefore, the utilization of task $\tau_1$ is less than or equal to 1. Assume $C_1/T_1 + C_2/T_2 = B_2 + \delta$. Since the optimal solution of computation time is $C^*$, when $C_1^*/T_1 + C_2^*/T_2 = B_2$, the system is schedulable and any increase of computation time will result in non-feasibility. We create a computation set $\{C_1', C_2'\}$, where

$$C_1' = C_1^*$$

$$C_2' = \delta T_2 + C_2^*.$$  

As $\delta > 0$, the fact that $C_2'$ is greater than $C_2^*$ causes the system to be infeasible. So we conclude that $C_1/T_1 + C_2/T_2 \leq B_2$.

Assume that tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$ are schedulable if

$$\forall_{m=1}^{k-1} \sum_{i=1}^{m} \frac{C_i}{T_i} \leq B_m$$

in a task set $\{\tau_1, \tau_2, \ldots, \tau_{k-1}\}$.

When the number of tasks is $k$, $B_k$ is the utilization bound of $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau_k\}$. For preemptive fixed-priority scheduling, the higher priority task can preempt the current processing task with lower priority. Hence tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$ are schedulable if

$$\forall_{m=1}^{k-1} \sum_{i=1}^{m} \frac{C_i}{T_i} \leq B_m$$

in a task set $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau_k\}$. For task $\tau_k$, when $\sum_{i=1}^{m} C_i/T_i \leq B_m$, it meets its deadline. Therefore, all tasks are schedulable.

Now we prove the necessity of the condition. Assume one priority subset has a higher utilization than its corresponding utilization bound and all tasks are still schedulable in any case. We know that $\forall_{m=1}^{k-1}, \sum_{i=1}^{m} C_i/T_i \leq B_m$ are satisfied if tasks
\( \tau_1, \tau_2, \ldots, \tau_{k-1} \) are schedulable. We suppose a priority \( k \) subset has a higher utilization, that is, \( \sum_{i=1}^{k} C_i/T_i = B_k + \delta \). As the optimal solution is \( C^{**} \), there exists a computation time set in which

\[ C_i' = C_i^{**} \]

\[ C_2' = C_2^{**} \]

\[ \vdots \]

\[ C_k' = \delta T_k + C_k^{**}. \]

Since \( \delta > 0 \), the computation time of task \( \tau_k \) increases. This causes task \( \tau_k \) to miss its deadline in the critical instant. So \( \sum_{i=1}^{k} C_i/T_i \leq B_k \) must be satisfied.

The condition \( \sum_{i=1}^{k} C_i/T_i \leq B_k \) is called the task schedulability constraint.

**Theorem 3.1** The subsystem utilization bound \( B_n \) for the schedulable task set \( \{\tau_1, \ldots, \tau_n\} \) can be achieved by the following procedure:

**Minimize**

\[ B_n = \sum_{i=1}^{n} \frac{C_i}{T_i} \]  

(3.22)

**subject to** \( \forall_{m=1}^{n} \)

\[ \sum_{i=1}^{m} \left[ \frac{D_m}{T_i} \right] C_i \geq D_m \]  

(3.23)

and \( \forall_{m=1}^{n-1} \)

\[ \sum_{i=1}^{m} \frac{C_i}{T_i} \leq B_m \]  

(3.24)

and \( \forall_{m=1}^{n} \)

\[ C_i \leq 0 \]

Proof. From Lemma 3.2, tasks \( \tau_1, \tau_2, \ldots, \tau_n \) are feasible because of Eq. 3.20. Equation 3.19 ensures the schedule is full utilization. Consequently, \( B_n \) is the subsystem utilization bound.
Theorem 3.2 When the individual subsystem bounds of a task set are $B_1, B_2, ..., B_n$, the system utilization bound $B(E)$ can be up to $B_n$ as $U_1 \leq B_1, U_2 \leq B_2, ..., U_{n-1} \leq B_{n-1}$.

The proof follows from Lemma 3.2.

3.3 Procedure of the Algorithm

We present the procedure to determine the system utilization bound. In the procedure, $B_i$ is calculated by Equation 3.18. Assume that there are $n$ tasks in a system.

Step 1: If all $C_1_{\text{max}}, C_2_{\text{max}}, ..., C_{n_{\text{max}}}$ is bounded
then perform the exact schedulability test by Equation 2.7
if feasible, then $B(E) = 100\%$ meaning the system is always feasible when $C_i \leq C_{i_{\text{max}}}$ and terminate.

Step 2: If any task does not pass the test with $C_1_{\text{min}}, C_2_{\text{min}}, ..., C_{n_{\text{min}}}$
Terminate (because the system is not feasible)

Step 3: For $i=1$ to $n$
Calculate the priority $i$ subsystem utilization bound $B_i$

Step 4: $B(E) = B_n$
3.4 Discussion and Comparison

In this section, we first take a look at the specific task sets which lead to the worst case utilization bound in Liu and Layland's algorithm. Then, Park's approach is compared to Liu's to demonstrate its improvement. Finally, exact feasible utilization bounds are compared to Park's utilization bounds. This section also discusses the cause of the pessimism in Liu and Layland's and Park's techniques.

Liu and Layland showed that a task set \( \{ \tau_1, ..., \tau_n \} \) has a relatively low utilization bound when the ratio between two request periods is less than 2. The worst case utilization bound occurs when

\[
\frac{T_n - T_i}{T_i} = 2^{\frac{n-i}{n}} - 1 \quad i = 1, ..., n - 1.
\]

A task set gains its worst case utilization bound when task's periodic times satisfy

\[
T_i = 2^{\frac{i-n}{n}} T_n \quad i = 0, 1, ..., n - 1.
\]

This specific task set has the exact utilization bound equal to \( n(2^{1/n} - 1) \).

Therefore, the utilization bounds of all task sets with 2 tasks are the utilization of a special task set that has corresponding task periods \{0.707, 1\}. As for a system with 4 tasks, the special task period set is \{0.595, 0.707, 0.8409, 1\} or a task set whose members are the multiple of the corresponding member in the special task period set, such as \{1.19, 1.414, 1.6818, 2\}.

Park has taken account of task periods to determine the utilization bound because task periods in hard real time systems depend on physical laws and/or requirements which may be known before the design. Park has proved in [Park 93] that his approach can obtain the same or better utilization bound of a real time system than Liu and Layland's algorithm.
Chapter 3. Exact Feasible Utilization Bound

Table 3.1: The comparison of Park’s and Liu’s algorithms.

<table>
<thead>
<tr>
<th>Period Set</th>
<th>(\frac{T_i}{T_1})</th>
<th>(\frac{T_i}{T_2})</th>
<th>(\frac{T_i}{T_3})</th>
<th>(U_2^*)</th>
<th>(U_3^*)</th>
<th>(U_4^*)</th>
<th>B(Park)</th>
<th>B(Liu)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{50,65,94,98}</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.8385</td>
<td>0.8100</td>
<td>0.8091</td>
<td>0.8091</td>
<td>0.8091</td>
</tr>
<tr>
<td>{35,63,78,79}</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.9111</td>
<td>0.8525</td>
<td>0.8459</td>
<td>0.8459</td>
<td>0.8332</td>
</tr>
<tr>
<td>{7,25,53,59}</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>0.9314</td>
<td>0.9421</td>
<td>0.8772</td>
<td>0.8772</td>
<td>0.8651</td>
</tr>
<tr>
<td>{5,49,107,483}</td>
<td>97</td>
<td>10</td>
<td>4</td>
<td>0.9837</td>
<td>0.9313</td>
<td>0.9447</td>
<td>0.9313</td>
<td>0.8651</td>
</tr>
</tbody>
</table>

First, the two algorithms lead to the same exact utilization bounds when the ratio between any two periods is less than 2 for a given task set \(\{\tau_1, \tau_2, \ldots, \tau_n\}\). The exact utilization bound can be calculated by Liu and Layland’s equation:

\[
B(\text{Liu}) = 1 + \frac{(T_n - T_1)(T_n - 2T_1)}{T_1T_n} + \sum_{i=2}^{n} \frac{(T_n - T_i)(T_{i-1} - T_i)}{T_iT_{i-1}}
\]

Since \(T_n - 2T_1 < 0\) (because \(T_n/T_1 \leq 2\) and \(T_{i-1} - T_i < 0\)), \(B(\text{Liu})\) will decrease with the increased task number, that is \(B_1(\text{Liu}) \geq B_2(\text{Liu}) \geq \ldots \geq B_n(\text{Liu})\). In Park’s algorithm, the individual task utilization bound \(U_i^*\) is equal to \(B_i(\text{Liu})\). Therefore the minimization in Park’s algorithm does not affect the accuracy of the system utilization bound. Both methods obtain the same results.

However, Park’s approach shows its advantage when the minimum ratio between task periods is greater than 2, as illustrated in Table 3.1. In the table, the exact utilization bound in the last column is obtained by multiplying the task periods that are less than half of the longest period to make all ratio of any two tasks to be less than 2, and then applying Liu’s exact utilization bound formula to the modified system.

Although the performance of Park’s technique is generally better than Liu’s, it is not able to obtain the exact utilization bound for the last task set in Table 3.1. When the system utilization bound is 0.9313, there exists idle time during the time \([0,483]\). \(U_4^*\) can not be the system utilization bound since the feasibilities of all tasks are not guaranteed.
Table 3.2: The comparison of Park’s and Our algorithms.

Table 3.2 shows the comparison between our technique and Park’s approach. As the individual utilization decreases with the increased number of tasks, items 1 and 4 show that our approach can obtain the same subsystem (individual) utilization bounds and system utilization bound as Park’s algorithm. When there is no decreasing function of the subsystem utilization bounds, Park’s algorithm gives a relatively poor result due to minimization as shown in item 8.

Items 2 and 3 indicate that the two approaches have different system utilization bounds when $B_4$ and $U^*_4$ are no longer the same. We take a close look at item 2 in Table 3.3. As the optimal solution of $U^*_4$ is computation time set \{5,0,580,10\}, the utilization of the first three tasks is $C_1/T_1 + C_2/T_2 + C_3/T_3 = 5/300 + 580/605 = 0.9753$, which exceeds the individual utilization bound $U^*_3$. To calculate $B_4$, constraint
Chapter 3. Exact Feasible Utilization Bound

Table 3.3: The comparison of computation times in Park's and our algorithms.

<table>
<thead>
<tr>
<th>Subset</th>
<th>EFUB</th>
<th>Park's</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$B$</td>
<td>$U^*$</td>
</tr>
<tr>
<td>2</td>
<td>0.8333</td>
<td>0.8333</td>
</tr>
<tr>
<td>3</td>
<td>0.8307</td>
<td>0.8307</td>
</tr>
<tr>
<td>4</td>
<td>0.9860</td>
<td>0.9837</td>
</tr>
</tbody>
</table>

Figure 3.1: The perspective of utilization bounds

$C_1/T_1 + C_2/T_2 + C_3/T_3 \leq B_3$ causes the feasible region (the set of all feasible points—a point is feasible if it satisfies all constraints) to shrink in linear programming and the optimal value varies. Consequently, $B_4 \geq U^*_4$.

3.5 Perspective on Utilization Bounds

In the perspective on utilization bounds shown in 3.1, the universal set includes all real time task sets being scheduled by fixed priority scheduling algorithms. This set is classified into 2 subsets: feasible task sets F and infeasible task sets I. The feasible
subset also consists of another subset, just feasible task set J. For a task set S, there exists an assignment of computation times which makes the system just feasible. The system utilization in S with this specific computation time set is the utilization bound of S. When the task set has a different computation set, the corresponding system will fall in either the infeasible area or the feasible area. If the utilization of a computation set is less than the utilization bound, the schedulability of the system including the computation set is guaranteed.

Liu and Layland’s theory only considers the utilization bound for a particular subset L of J, in which a specific task set represents all task sets with the same number of tasks, no matter what their periodic times are. Park’s approach covers a larger area W where individual task’s periodic times are taken into consideration but the system utilization bounds of some task sets are forced to be the values of their corresponding subsystems. For example, the utilization bounds of the task sets with periodic sets \(\{19,23,39,105\}\) and \(\{19,23,39,105,220\}\) are determined by a task set with periodic sets \(\{19,23,39\}\).

From the above discussion, our algorithm is an extension of both Liu and Park’s algorithm. The exact feasible utilization bound for a subset E of J covers L and W.

### 3.6 Consideration on Some Known Computation Times

In the preceding section, we have discussed how to estimate system schedulabilities in a real time system with unknown computation times in the design phase. Our approach can also be used in the successive phases of software development lifecycle. After a few tasks have been implemented in the coding phase or redesigned during maintenance, some tasks' execution times are known. Since we adopt a range for all computation times, the feasible region is relatively large and the calculated
utilization bound is relatively rough. If some computation times are known, the system utilization bound may get larger.

The procedure to estimate system utilization bound with some known computation times is different from the general one in that $C_{\text{imin}} = C_{\text{imax}} = C_i$ in steps 1 and 2 (of our procedure in Section 3.3) and the priority $i$ utilization bound is set to 100% without calculation.

The increased system utilization bound may be achieved by using precise computation times, which is demonstrated in the following example. Assume a period set in a task set is $\{14,44,50,63\}$. We know the system utilization bound reaches 0.7931 when the computation time set is $\{0,6,13,25\}$ under the condition $U_1 \leq 1$, $U_2 \leq 0.9610$, and $U_3 \leq 0.8792$. It is reasonable to assume that the system utilization can be over 0.7931 because the computation time of task $\tau_1$ can not be 0. If $C_1$ is 5, the system utilization bound is increased to 0.8393. This utilization bound also belongs to the subset $J$ in Figure 3.3 but is not covered by subset $E$.

### 3.7 Environmental Issues

Now we will examine how our utilization analysis approach can be applied to real-world real time systems.

#### 3.7.1 Sporadic Tasks

Sporadic tasks play an important role in real time systems. A sporadic task is released by an event originating either from another task or from the environment of the system through interrupts. The common idea to deal with sporadic tasks is the sporadic server algorithm [Tilborg 90]. The sporadic server algorithm creates a periodic task $\tau_{ss}$ that is assigned:
Chapter 3. Exact Feasible Utilization Bound

1. a computation budget $C_{ss}$,

2. a replenishment period $T_{ss}$ which is the minimum event inter-arrival time,

3. a priority $P_{ss}$.

When the interrupt event occurs, task $\tau_{ss}$ is scheduled with priority $P_{ss}$ until the task completes execution or the computation budget has been used up. If the request has not been serviced completely, task $\tau_{ss}$ will be resumed in $T_{ss} - C_{ss}$ time units. Otherwise, task $\tau_{ss}$ is in the idle state in which the computation budget is replenished. The following request will not be scheduled during $T_{ss} - C_{ss}$ time units.

By applying the sporadic server algorithm, a sporadic task is treated like a periodic task and several sporadic servers can be defined at different priority levels to handle different aperiodic events. The discussed scheduling algorithm and schedulability analysis can be carried out in a system with both periodic and aperiodic tasks.

3.7.2 Task Synchronization

Priority-based preemptive scheduling assumes that a higher priority task can preempt a lower priority task being run with no penalties in time. The problem of blocking is caused by the synchronization of tasks that share physical or logical resources when a process is suspended waiting for a low-priority task to complete some required computation.

Synchronization mechanisms may lead to an unbounded duration of priority inversion, multiple blocking, and mutual deadlock because of the nature of task synchronization including semaphores, locks, monitors, and rendezvous which is necessary to protect the consistency of shared data and the correctness of computation. The run time behavior of each task becomes unpredictable.
In this thesis the priority ceiling protocol [Rajkumar 91] is considered because it addresses the above problems. In this protocol, besides the static priorities of each task, each critical section has a static ceiling priority value which is the maximum priority of the tasks accessing the critical section. The underlying idea of this protocol is to ensure that when a task τ preempts the critical section $S_1$ of another task and executes its own critical section, the priority of task τ is guaranteed to be higher than the priority of $S_1$. This idea is realized by first assigning a priority ceiling to each semaphore, which is equal to the highest priority task that may use this semaphore. Then task τ starts a new section only if τ’s priority is higher than an priority ceilings of all the semaphores locked by tasks other than τ. The blocking is bounded to the maximum critical sections, $b_k$, $1 \leq k \leq n$. Each task but one with the least priority may experience the blocking if they synchronized with other tasks. The blocking of task $τ_i$ is equal to maximum worst case execution time of all lower priority level tasks in the same critical section. Therefore when task $τ_i$ has to access the critical section $S_1$ and tasks $τ_{i-1}$, $τ_{i+1}$ and $τ_{i+3}$ also access the same section, task $τ_i$ can be blocked by tasks $τ_{i+1}$ and $τ_{i+3}$. The blocking time $C_{bi}$ is the maximum worst case execution time of $τ_{i+1}$ and $τ_{i+3}$ in $S_1$.

In the schedulability test, the blocking time can be treated as part of the worst case computation time, so the new computation time becomes

$$C_i' = C_i + C_{bi}. \quad (3.25)$$

### 3.7.3 Scheduler Overhead

The context switch and queue manipulation time has not been considered yet in a scheduler, which may be of significance and affects the schedulability as the systems become more and more complex. We consider interrupt event-driven scheduling...
which is quite popular and can be adopted by schedulers using rate monotonic and deadline monotonic algorithms in which periodic tasks are activated by system timers [Katcher 93]. In this scheduling, every interrupt is responded to and all tasks are triggered by external interrupts. If the arriving task has a higher priority, it preempts the current task. Otherwise, the arriving task is rescheduled and put in the ready queue and the current task continues running after a brief interference. Hence the overhead for a task with higher priority than the current one is $C_{\text{preempt}}$ and for a task with lower priority, it is $C_{\text{nonpreempt}}$. Besides that, each task has an exit process time.

$$C_{\text{preempt}} = C_{\text{int}} + C_{\text{store}} + C_{\text{load}}$$  \hspace{1cm} (3.26)

$$C_{\text{exit}} = C_{\text{trap}} + C_{\text{load}}$$ \hspace{1cm} (3.27)

$$C_{\text{nonpreempt}} = C_{\text{int}} + C_{\text{sched}} + C_{\text{resume}}$$ \hspace{1cm} (3.28)

where $C_{\text{int}}$ is the time to handle an interrupt, $C_{\text{store}}$ represents the time to save the state of the active task to a task control block, $C_{\text{load}}$ is the time to load the new active task state, and $C_{\text{trap}}$ equals to the time to handle the trap generated by normal completion of a task, $C_{\text{sched}}$ is the time to execute the scheduling code to determine the next task to run, $C_{\text{resume}}$ represents the time to return to the previously active task when a preemption does not occur. As the execution time of system overhead is deterministic in hard real time systems, the time of $C_{\text{preempt}}$, $C_{\text{nonpreempt}}$, and $C_{\text{exit}}$ are constant values. So each task will suffer the overhead $C_o$ up to either $C_{\text{preempt}} + C_{\text{exit}}$ or $C_{\text{nonpreempt}} + C_{\text{exit}}$. The effect of the overhead can be treated as

$$C_i' = C_i + C_o.$$ \hspace{1cm} (3.29)
3.7.4 The Effect on Subsystem Utilization Bound

The blocking and overhead time effect the calculation of the subsystem utilization bound as follows:

**Theorem 3.3** The subsystem utilization bound \( B_n \) for the schedulable task set \( \{\tau_1, ..., \tau_n\} \) can be achieved by the following procedure:

Minimize

\[
B_n = \sum_{i=1}^{n} \frac{C'_i}{T_i} \tag{3.30}
\]

subject to \( \forall_{m=1}^{n} \)

\[
\sum_{i=1}^{n} \left[ \frac{D_m}{T_i} \right] C'_i \geq D_m \tag{3.31}
\]

and \( \forall_{m=1}^{n-1} \)

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq B_m \tag{3.32}
\]

and \( \forall_{m=1}^{n} \)

\[
C'_i = C_i + C_{bi} + C_o
\]

\[
C_i \leq C_{bi} + C_o
\]
Chapter 4

Case Study

This chapter discusses two real time systems to demonstrate how to apply the utilization bound analysis technique developed in the previous chapter in the design phase. The two systems are respectively a mine pump control system and a hydraulic motion platform simulator. We follow the procedure of software engineering to identify and analyze functional and timing requirements, then decompose the system into subsystems and structure subsystems into concurrent tasks. Furthermore, we construct a system model that determines each task's period and deadline according to the timing requirements, and then assign a unique priority to each task. With the system model, we apply the exact utilization bound approach to calculate the utilization bound.

Throughout the case study, several practical issues in hard real time systems are addressed including:

- **Non-critical tasks.** Non-critical tasks often exist to perform housekeeping work in hard real time system. When we design this kind of task, we must ensure the schedulability of critical tasks and assign a reasonable computation time to non-critical tasks.

- **Non-periodic tasks.**

- **Tasks without a stringent period.** When the minimum arrival rates of some tasks is in a range, such as tasks interacting with operators, we can assign their periods as a multiple of another task's period. In this case, the CPU resource
can be fully exploited and schedulability analysis is simplified.

• Task deadline much less than its period. An extended approach of the exact utilization bound is suggested to analyze the utilization budget of a task with very low maximum task utilization.

4.1 Mine Pump and Control System

The mine pump and control system is intended to monitor the levels of carbon monoxide, methane and airflow and to pump water out of a sump if percolating water is high and to set an alarm if necessary [Joseph 86]. The system is depicted in Figure 4.1. The functional requirements are as follows:

1. Water is pumped out of the mine if water is above the high level D under the condition that the methane level is below a critical level.

2. The pump is switched off when the water goes down below the low level E and the level of methane exceeds the critical value.

3. The mine must be evacuated within one hour of the pump failing.

4. An alarm is raised and operator is informed within one second when any level of carbon monoxide, methane and airflow exceeds the corresponding critical value. These data are sampled periodically.

5. Human operators can control the operation of the pump if the water is between the low and high water levels under the condition that the methane level is below a critical level.

6. The supervisor can operate the pump under the condition that the methane level is below a critical level. Both supervisor and operator can check the current
7. Data from all sensors and a record of the operation of the pump must be logged for later analysis.

In addition the timing requirements of nonfunctional requirements are also listed:

1. Emergency shut down following a high methane value reading: the deadline is 30 milliseconds.
2. Recognition of monitor failure and pump shut down: the deadline is 65 milliseconds.

3. Turning the pump on again when it is safe: the deadline is 100 milliseconds.

4. Turning the pump on (if safe) when the water is high: the deadline is 100 milliseconds.

5. Turning the pump off when the low water level has been reached: the deadline is 75 milliseconds.

6. Signaling an alarm if any environmental condition warrants it: the deadline is 50 milliseconds.

Since the software design methods of a real time system [Gomaa 93] is not a concern in this thesis, the design procedure is not detailed and only the design of functional requirements are outlined. HRT-ROOM (Hard Real Time-Real Time Object Oriented Modeling) is used to design this real time system [Joseph 86]. Based on HRT-ROOM the mine pump and control system can be decomposed into four subsystems:

- the pump controller
- the environmental monitors(for airflow, methane and carbon monoxide)
- the data logging subsystem
- the operator's subsystem

Figure 4.2 shows the task modules and communications among the tasks. Then we will construct the system model based on the timing requirements and task modules.
Figure 4.2: Design for the mine pump and control system
4.1.1 System Model

To meet both the functional and non-functional requirements, the system can be modelled as a set of concurrent tasks and shared objects through which tasks communicate with each other. Tasks may be periodic or sporadic. A periodic task is released by a timer event and a sporadic task is activated by external interrupt or another task.

In the mine pump and control system, tasks Methane Monitor (MM), Air Monitor (AM), CO Monitor (CM), and Safety Checker (SC) are periodic tasks. The first three tasks sample data periodically from the corresponding sensors and process the data because most of signal processing algorithms are based on evenly spaced samplings.

The periods of the tasks are decided according to the timing requirements. Since the emergency shut down must be performed within 30 milliseconds when the level of methane is high, the worst case is considered when the level of methane exceeds the threshold just after the sampling. Therefore the sum of sampling interval and response time is set to be 30\(ms\). We can either set both the period and deadline to 15\(ms\) or period 20\(ms\) and deadline 10\(ms\) by reducing the task service times. Because the task MM has a relative high priority in the system, we adopt the latter assignment. Tasks AM and CM are designed with both the period and deadline of 25\(ms\) due to the fifth timing requirement. Task SC detecting the failure of pump and monitors within 65\(ms\) is assigned a period of 35\(ms\) and deadline of 30\(ms\).

Tasks HS and LS are sporadic tasks triggered by interrupts. An extra task Interrupt Handler (IH) is created to respond to the hardware interrupts and invoke the corresponding task. Task IH can be regarded as system overhead to other tasks as its function is to make the system reschedule between the running task and ready tasks.
Table 4.1: Mine control tasks and protected shared objects.

The equations from 3.26 to 3.29 are suitable for task IH to calculate the system overhead of tasks HS and LS. The period of task IH depends on period of tasks HS and LS. The minimal inter-arrival time of tasks HS and LS is related to the rate of water percolating that comes from the previous records. Suppose that it takes 10 minutes for water to raise from the low level to high level in the sump, that is, the minimum inter-arrival interval of external interrupt is 10 minutes. The periods of HS and LS are predicted to be 10,000ms. Their deadlines are 100ms and 75ms respectively.

Tasks Operator (OP) and Logging_Task (LT) do not have associated deadlines and are considered as non-critical tasks which are discussed later.

After task periods and deadlines are determined, the priorities are allocated to them. In this case both rate monotonic and deadline monotonic algorithms will obtain the same priority order in Table 4.1. The table also lists all protected shared objects (PSO) in the system and their ceiling priorities.
4.1.2 Non-Critical Tasks

In hard real time systems, the consequences of missing deadlines with non-critical tasks are not serious. Task OP is an aperiodic task with no specified period and provides the service to display system status and to control the pump manually by an operator. This task does not have timing requirement and will not cause a catastrophe if its deadline is missed. This task can be designed as a background job that activates when the CPU is in the idle state. In this case, the utilization bound cannot be up to 1. Since the operation in task OP is simple, a 5 percent utilization margin will satisfy its usage. Therefore when we calculate utilization bound, we do not include task OP. However, the utilization bound for the remaining tasks is limited to 95%.

Task LT is different from task OP in that task LT must record the data before the buffer storing the sampling data is overfilled, that is, there exists a hard deadline. This task can be deferred if sufficient buffer space is provided. Thus, task LT is designed as a periodic task with two alternately used blocks of data buffers. The size of each buffer is determined by the data structure of sampling data. The priority of task LT is lower than any other critical tasks so that the execution of task LT does not affect the schedulability of other critical tasks. In this system, both the period and deadline of task LT are assigned to be 600ms.

Another reason to give task LT a low priority is the concern of system overload. The periods of tasks HS and LS result from heuristics. It is possible that the percolating rate is higher than anticipated. Under this special circumstance, there are two requests of task HS and LS during the minimum inter-arrival interval and the execution of tasks HS and LS may exceed the predicted time. To keep the safety of the mine, we ensure that these two tasks can take some time from task LT. After the critical instant, new periods of task HS and LS are created and the system's
Table 4.2: The task set of the mine control system.

performance should be re-checked.

4.1.3 Schedulability Analysis

Before the schedulability is analyzed, all tasks are re-organized by merging two tasks with the same periods into one to simplify the analysis. As tasks AM and CM have the same periods and deadlines, they are combined into one. Tasks HS and LS emerge with the period of 10000ms and deadline of 75ms for simplicity. The new task set is listed in Table 4.2.

Applying our exact utilization bound approach to this task set, we obtain an interesting result with the following subsystem utilization bounds:

\[
B_1 = 0.5000 \\
B_2 = 0.6667 \\
B_3 = 0.7857 \\
B_4 = 0.0075 \\
B_5 = 0.8753
\]
The utilization bound of the priority 4 subsystem $B_4$ shows that more than 99% CPU resource can not be used for the execution of the first four tasks. That is not practical. An alternative approach is required to circumvent the problem.

The reason for the low utilization bound $B_4$ is that the maximum task utilization of task $r_4$, equal to $D_4/T_4$, is much less than 1. That causes the $B_4$ to be equal to or less than the maximum utilization of task $r_4$.

To explain this observation, in the linear programming to calculate utilization bound, the objective function of the minimization is

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{C_4}{T_4}$$

subject to

$$\left[ \frac{T_1}{T_1} \right] C_1 + \left[ \frac{T_4}{T_2} \right] C_2 + \left[ \frac{T_4}{T_3} \right] C_3 + C_4 = D_4$$

and

$$T_1 < T_2 < T_3 < T_4.$$ 

Then, we have

$$U = \frac{T_4 C_1}{T_4} + \frac{T_4 C_2}{T_4} + \frac{T_4 C_3}{T_4} + \frac{C_4}{T_4}$$

$$\leq \frac{\left[ \frac{T_4}{T_1} \right] C_1}{T_4} + \frac{\left[ \frac{T_4}{T_2} \right] C_2}{T_4} + \frac{\left[ \frac{T_4}{T_3} \right] C_3}{T_4} + \frac{C_4}{T_4}$$

$$= \frac{D_4}{T_4}.$$ 

Consequently, we can conclude that the utilization bound approach is suitable for a system in which the maximum utilization of each task is close to 1.

As a solution to this problem, we assume that the period of task $r_4$ is 75 in order to estimate the system utilization bound. As we shorten the period, the system load
increases because the utilization of task $\tau_4$ changes from $C_4/T_4$ to $C_4/D_4$. In this case, the utilization bounds of all subsystems are

\begin{align*}
B_1 &= 0.5000 \\
B_2 &= 0.6667 \\
B_3 &= 0.7857 \\
B_4 &= 0.8762 \\
B_5 &= 0.9929
\end{align*}

Let us discuss the effects of the assumption that $\tau_4$'s period can be decreased from $10,000$ to $75$. We distinguish the two task sets as $S_1$ and $S_2$. Suppose the system is schedulable when $T_4 = 75$ with any computation set $\{C_{11}, C_{21}, C_{31}, C_{41}, C_{51}\}$ and $U_1 \leq 0.5, U_2 \leq 0.6667, U_3 \leq 0.7857, U_4 \leq 0.8762, U_5 \leq 0.9929$. Since tasks $\tau_1$, $\tau_2$, and $\tau_3$ have higher priorities than task $\tau_4$'s, their schedule can not be affected by the request of task $\tau_4$, regardless of $\tau_4$'s period. Task $\tau_4$ is guaranteed to be schedulable when the period is $75$ and it is still schedulable when its period is increased to $10,000$. So far, the scheduler can ensure that the first four tasks meet their deadlines. The only difference is that the two task sets have different utilization when $C_{41}$ is not zero because $U_{S_1} - U_{S_2} = C_{41}(1/10,000 - 1/75)$. As a result, the impact on task $\tau_5$ is not favorable because the utilization of task $\tau_4$ increases dramatically as its period is decreased. When $C_{11} = 10, C_{21} = C_{31} = 0, C_{41} = 75, and C_{51} = 240$, the utilization of the system is $0.9075$ which is less than bound $B_5$. Unfortunately, the response time of task $\tau_5$ is $615$ which is greater than its deadline. The schedulability of task $\tau_5$ is no longer guaranteed. That means that $T_4 = 75$ can not be used to calculate the subsystem utilization bound when the subsystem includes task $\tau_4$. Therefore, to calculate the $B_5$, we add in one more constraint for task $\tau_4$, $U_{\tau_4} \leq D_4/T_4$. 
Chapter 4. Case Study

We summarize our procedure to calculate the utilization bound of a system in which a task's deadline is much less than its period:

1. Calculate the utilization bound of $B_1$, $B_2$, and $B_3$.

2. Calculate the utilization bound of $B_4$ with $T_4 = D_4 = 75$.

3. Calculate the utilization bound of $B_5$ with $C_4/T_4 \leq 0.0075$ and $U_4 \leq B_4$ where $T_4 = 10,000$.

Finally, each task can be assigned an execution time budget based on the utilization bounds: $U_1 \leq 0.5$; $U_2 \leq B_1 = 0.6667$; $U_3 \leq B_2 = 0.7857$; $U_4 \leq B_3 = 0.8753$ and $U_{r_4} \leq 0.0075$; and $U_5 \leq B_5 = 0.8753$.

4.2 Hydraulic Motion Platform Simulator System

Figure 4.3 shows a hydraulic motion platform simulator system implemented with VxWorks real time operating system on a SPARC target board. This system has been developed by the Robotics Laboratory in the Department of Electrical and Computer Engineering at UBC. The system controls the hydraulic platform through console and/or joysticks with the following functional requirements.

Figure 4.3: The system diagram of the hydraulic motion platform simulator system
Figure 4.4: Design for the hydraulic motion platform simulator system

1. The system controls the hydraulic valves on the motion simulator via D/A converters and detects the position of the platform through the digitized signals of the magnetic sensors.

2. An operator controls the hydraulic motion platform by joystick.

3. An operator can send commands to the system through console keyboard.

The timing requirements are

1. The A/D converter rate is up to 400Hz.

2. The human commands must be responded to within 50ms.

The system consists of four periodic tasks: interrupt service request handler, motion controller, trajectory planner, and user controller.

1. Interrupt handler (IH) is triggered by a hardware timer to invoke periodic tasks according their time periods. The execution time of IH is regarded as system overhead.
2. Motion controller (MC) samples the current platform position, compares the data to the required one, and updates hydraulic valve parameters.

3. Trajectory planner (TP) samples joysticks and generates the parameters of the hydraulic valves from the sampling data. It also converts commands from the console keyboard to the specific parameters.

4. User (UC) validates commands.

4.2.1 System Model

The period of task MC depends on the A/D convert rate. If the task period is less than that of A/D convert, task MC overwrites the previous data. The period of task MC is assigned as 2.5ms.

The response time of a signal from the joystick must be less than 50ms. We set the period of task TP to a multiple of the period of task MC. Task UC is a non-critical task which does not have an associated period. As the keyboard is polled by task UC, we assign the period to be 20ms by using short command length with one or two letters.

The priority of each task is assigned using the rate monotonic scheduling algorithm. The periods of task TP and UC are the same. Task TP is assigned a higher priority because task UC is a non-critical task. To communicate among the tasks, protected shared data control parameter (CP) and control command (CC) are used. All task parameters are listed in Table 4.3.

4.2.2 Real Time Systems with Harmonic Periods

In the hydraulic motion platform simulator system, there is no constant periodic requirement for task TP. We choose its period as the multiple of task MC because
1. the utilization can be up to 1 when all task periods are harmonically related.

2. the computation times of the tasks can be budgeted linearly.

The first property has been noticed by Liu and Layland [Liu 73]. As an example of the two properties, consider that for linear programming to calculate utilization bound, the objective function of the minimization is

$$ U = \frac{C_1}{T_1} + \frac{C_2}{T_2} $$

subject to

$$ \left[ \frac{T_2}{T_1} \right] C_1 + C_2 = D_2 $$

and

$$ T_1 < T_2. $$

When $T_2$ is the multiple of $T_1$, the ceiling of $T_2/T_1$ is equal to $T_2/T_1$. Then, we have

$$ U = \frac{C_1}{T_1} + \frac{C_2}{T_2} $$

$$ = \frac{T_2 C_1}{T_1 T_2} + \frac{C_2}{T_2} $$
Chapter 4. Case Study

The utilization reaches 1 when deadline $D_2$ is equal to its corresponding period $T_2$. So it is possible to design a real time system with harmonic periods to use all CPU resources and still guarantee feasibility.

When the total executable interval is $D_2$ for task MC and TP, the computation time distribution between tasks MC and TP is a linear equation

$$
\frac{T_2}{T_1} C_1 + C_2 = D_2.
$$

The system time relationship for all tasks is

$$
\frac{T_2}{T_1} C_1 + C_2 + C_3 = D_3.
$$

In this case, it is convenient for developers to budget task computation times and this system is schedulable when

$$
\frac{T_2}{T_1} C_1 + C_2 + C_3 \leq D_3.
$$
Chapter 5

Conclusions and Future Work

5.1 Summary of Results

In this thesis, we have proposed a technique to calculate the exact feasible utilization bound of a task set in a real-time system. Liu and Layland's approach obtains the worst case utilization bound without considering any task information, while Park's algorithm accomplished the relatively accurate schedulability test by taking into account task information. Our approach enhances Park's algorithm by obtaining the exact utilization bound for a task set.

While algorithmic-based scheduling schemes are developed to order the use of system resources, they also provide a means to predict the worst-case behavior of a system. The pioneering work of Liu and Layland offers the framework for scheduling periodic tasks and computing the worst case utilization bound for a real time system. However, the utilization bound provided by the approach is substantially less than the average case behavior in practice. Moreover, this technique is only applicable to the system that assigns task priorities by the rate monotonic scheduling algorithm which is optimal only when the task deadlines are equal to their corresponding periods. Instead, the deadline monotonic scheduling algorithm is proved to be the optimal algorithm for scheduling a real time system with shorter task deadlines than their periods. Therefore a schedulability testing method is desired for the deadline monotonic scheduling algorithm.
The goal of our technique is to determine the exact utilization bound before detailed information of task computation times in a task set is known. While linear programming is used in our algorithm, the constraints are classified into three groups:

1. Full utilization constraints: guarantee that there is no idle time in the processor within the first task deadlines.

2. Task schedulability constraints: guarantee that tasks in a subsystem are schedulable.

3. Computation time constraints.

Therefore, the approach can obtain the exact utilization bound by accommodating both full utilization and task schedulability conditions. In addition, the approach has the following features:

1. Task deadlines can be arbitrary values.

2. Task priorities can be assigned by rate monotonic algorithm, deadline monotonic algorithm, or any other algorithms.

This algorithm can obtain better results of utilization bound than either Liu's or Park's algorithms do. If the utilization bounds decrease as the number of tasks increase, the utilization bounds estimated by the three methods are identical. In other cases, Liu's utilization bounds are much less than those calculated using our method. Compared to Park's algorithm, our approach finds utilization bounds in a feasible region satisfying full utilization and task schedulability by applying linear programming. Because this region in evaluating the optimum does not exceed the region in Park's algorithm, the minimum value in our region is equal to or greater
than that in Park's region. Therefore our approach gains a more accurate utilization bound than Park's.

In addition, this thesis investigates the application of utilization bound analysis to the software design phase. Our findings indicate that utilization bound analysis provides new and useful information for the design. It is also complementary to simulation-based analysis. In the course of applying our technique to two sample systems, we show how the approach can be adapted to handle the following situations:

1. Hard real time tasks have higher priorities than non-critical tasks do. Some non-critical tasks are designed as background jobs with a certain processor utilization.

2. Sporadic tasks are designed as periodic tasks with the periods evaluated as the minimum arrive rates of sporadic external events.

3. When a task period is not strict, its period can be made to be a multiple of another task's period, which can increase the overall system utilization bound.

5.2 Future Work

Although our algorithm can achieve the exact utilization bound for fixed-priority scheduling algorithms, there are still improvements left to be made.

5.2.1 Utilization Bound of a Real Time System with Shorter Deadlines than Periods

As task deadlines are shorter than the periods, our approach is still applicable because a task is schedulable when its first deadline is met after a critical instant. But a utilization bound deteriorates when the factor between the deadline and the period
is extremely small. The utilization bound test is not applicable to these systems. Although an alternative approach has been provided in the aforementioned case study, further analysis of performance is required.

5.2.2 Utilization Bound of Deferred Real Time Systems

When task deadlines are greater than their periods, a task may meet its first deadline after a critical instant but this does not ensure the task is still schedulable. A level-i busy period has been proposed by Lehoczky [Lehoczky 90] for schedulability test:

A level-i busy period is a time interval $[A,B]$ within which jobs of priority $i$ or higher are processed throughout $[A,B]$ but no jobs of level $i$ or higher are processed in $(c-c,a)$ or $(b,b+\epsilon)$ for sufficiently small $\epsilon > 0$.

For the interval $[A,B]$, value $A$ can be set to 0 at the critical instant, but $B$ is indeterminable without the knowledge of computation times. The task schedulability constraints must be revisited to ensure that those constraints are still valid.

5.2.3 Soft Real Time Systems

Unlike in hard real time systems, the value of a task result decreases gradually after the deadline expires in soft real time systems. Therefore, software design of soft real time systems place more emphasis on system average performance. Validity of the utilization test is yet determined and extended.
Bibliography


Bibliography


