NANYANG TECHNOLOGICAL UNIVERSITY

SCE13-0026

Micrium μC/OS-III++

Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Computer Engineering of the Nanyang Technological University

by

Do Nhat Minh

School of Computer Engineering

March 14, 2014

Abstract

Real-time operating systems play an important role in time and safety-critical software systems used in many fields, such as avionics, automotives and defense applications. μC/OS-III is an open-source real-time operating system which aims to be used on embedded devices with restricted resources. μC/OS-III has many useful features; however, it does not have better implementations proven in more recent advances in real-time systems research.

In this thesis, Earliest Deadline First scheduling algorithm and Stack Resource Policy for time-guaranteed resource sharing were implemented.

Acknowledgements

I am particularly grateful to my supervisor, Dr. Arvind Easwaran from the School of Computer Engineering, for being extremely supportive of my project choice and for giving invaluable advice during the course of this final year project.

I would like to thank Tran Ngoc Khanh Thy for the continued emotional support during the course of this final year project.

Contents

Chapter 1

Introduction

An operating system is a collection of software, or software components, which can be characterized as serving the following purposes, (1) interfacing with the underlying hardware to provide convenient abstractions for application programmers, and (2) managing the programs running on the system so that misbehaving programs do not impede others (Witchel, E. , 2009).

A real-time operating system is an operating system which must adhere to a real-time constraint. In such a system, the timeliness of the results from programs are as important as the correctness of such solutions. The system risks catastrophic failures if deadlines are missed. Some important applications for real-time operating systems are in avionics and automotives, where missing task deadlines leads to lives lost.

μC/OS-III is an open-source priority-based pre-emptive real-time multitasking operating system for embedded systems. It has many useful features which aims to cut development time (Labrosse, J. J. , 2010, pp. 27-31).

However, μC/OS-III does not implement deadline management but defer this task to the programmer. Moreover, uC/OS-III has no facility to specify nor keep track of the deadlines for running tasks of the system.

The goal of this thesis is to implement better algorithms for task scheduling and resource sharing for μC/OS-III. For task scheduling, Earliest Deadline First scheduling is implemented, and for resource sharing, Stack Resource Policy is implemented.

Chapter 2

Background

2.1 Scheduling Algorithms

A scheduling algorithm is an algorithm to determine the ordering of task executions in order to maximize resource utilization, while satisfying safety and correctness (Liu, C. L. and Layland, J. W. , 1973).

A real-time system consists of a number of tasks, each of which has a deadline and a period. Tasks must be completed before their deadlines or risk catastrophic consequences, e.g. a plane might crash if the task that sends sensor failure status misses deadlines. Task period is the interval between release times of instances of a task. Request rate is the reciprocal of task period. A task set is a collection of tasks to be scheduled. A task set is feasible when there exists an ordering where no deadline is missed.

A fixed priority scheduling algorithm is a scheduling algorithm where task orderings are based on statically assigned priorities of the tasks. The rate monotonic priority assignment assumes that a task with higher request rate is assigned a higher priority. C. L. Liu and J. W. Layland in their seminal paper have proven that for fixed priority scheduling, if there exists a feasible assignment, the rate monotonic assignment is also feasible (Liu, C. L. and Layland, J. W. , 1973). In other words, rate monotonic scheduling algorithm (RMS) is optimal.

CPU utilization for a task is the ratio between the time spent in execution and its period. CPU utilization for a task set is the summation of CPU utilization of

each task in the set.

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

where C_i is execution time and T_i is period for task i, n is the number of tasks in the task set and U is CPU utilization.

It is proven that CPU utilization for a task set in fixed priority scheduling must be kept below an upper bound in order to guarantee feasibility. This upper bound is

$$U_{RMS} \leq n(2^{\frac{1}{n}} - 1)$$

where n is the number of tasks in that task set and U_{RMS} is CPU utilization (Liu, C. L. and Layland, J. W. , 1973).

As n tends towards infinity, this expression will tend towards

$$\lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \approx 0.693147\ldots$$

As a rule of thumb, a task set is feasible under fixed priority scheduling when its CPU utilization is below 69.3%. The other 30.7% of CPU time can be reserved for other non real-time tasks.

However, this upper bound is pessimistic. It has been shown that a randomly generated task set will meet all deadlines when utilization is below 85% if exact task deadlines and periods are known, which might be difficult to achieve (Lehoczky, J.; Sha, L. and Ding, Y. , 1989).

A deadline driven scheduling algorithm is a scheduling algorithm where task orderings are based on deadlines of task instances. Under this scheme, an instance of a task is assigned highest priority if its deadline is nearest, while an instance of a task with a deadline that is farthest is assigned the lowest priority. This scheduling algorithm is also known as earliest deadline first scheduling algorithm (EDF). The utilization bound for EDF is

$$U_{EDF} \leq 1$$

where U_{EDF} is the CPU utilization (Liu, C. L. and Layland, J. W. , 1973).

From the utilization bounds for RMS and EDF, it is trivial to see that EDF guarantees all deadlines of a task set at a higher load than RMS. Therefore, EDF is more desirable from a resource utilization standpoint.

2.2 Implementations of Earliest Deadline First scheduling algorithms

2.3 Resource Sharing with Mutexes

2.4 Implementations of Priority Ceiling Protocol for Resource Sharing

2.5 Stack Resource Policy

Chapter 3

Overview of Micrium μC/OS-III

3.1 Task Model

Tasks in μC/OS-III are implemented as normal C functions with their own accompanying Task Control Blocks and stacks. However, unlike normal C functions, tasks are not allowed to return (Labrosse, J. J. , 2010, p. 83).

Task Control Block (TCB) is a C `struct` which holds necessary task-related information on which the whole of μC/OS-III depends in order to function properly. The information contained in a TCB includes a pointer to the top of stack, a pointer to the C function underlying this task, the current state of this task, the priority of this task and many more.

There are two type of tasks, namely run-to-completion and infinite loop. Run-to-completion tasks must call `OSTaskDel()` at the end of the function (Labrosse, J. J. , 2010, p. 84).

3.2 Scheduling Algorithm

μC/OS-III has a priority-based, preemptive scheduler. (Labrosse, J. J. , 2010, p. 141).

Priority-based

Each task are assigned a static priority. The kernel schedule them based on their priorities.

Preemptive

Higher priority tasks can preempt lower priority tasks, which means that during execution of a low priority task, if a high priority task is ready, the low priority task may be suspended so as to give CPU time to the high priority task.

3.3 Resource Sharing

Chapter 4

Modifications

4.1 Earliest Deadline First Scheduling

Earliest Deadline First Scheduling (EDF) is a dynamic scheduling algorithm.

4.1.1 Notes

EDF does not care about the relative deadline of the task but cares about its absolute deadline and its TCB.

The task spawner which spawns jobs for recurrent tasks cares about their relative deadlines and periodicities.

Because of task periodicities, the task spawner must allocate memory dynamically (for TCBs, stacks) to create job instances for tasks.

The OSRdyHeap is used to keep track of ready tasks with deadlines. The heap is a min heap w.r.t absolute deadlines.

The SpawnerHeap is used to keep track of which task to spawn next; this heap is also a min heap w.r.t. absolute spawn time.

$\Rightarrow$ Need for coarser time tick management: running spawner on every tick is expensive $\rightarrow$ cannot make guarantees about timeliness of task spawning.

Dynamic memory allocation problem:

Let the user allocate a block of memory for TCBs (this leads to problems with how tasks can communicate)

Bibliography

Witchel, E. (2009) *CS372 Operating Systems*. Retrieved from http://www.cs.utexas.edu/users/witchel/372/lectures/01.OSHistory.pdf on 2014/03/10.

Labrosse, J. J. (2010) *μC/OS-III: The Real-Time Kernel*. Retrieved from http://micrium.com/books/ucosiii/ti-lm3s9b92/

Liu, C. L. and Layland, J. W. (1973) *Scheduling Algorithm for multiprogramming in a Hard-Real-Time Environment*. Journal of ACM, 1973, vol 20, no 1, pp. 46-61.

Lehoczky, J.; Sha, L. and Ding, Y. (1989) *The rate monotonic scheduling algorithm: exact characterization and average case behavior*. IEEE Real-Time Systems Symposium, pp. 166-171.