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 10, 2014

Abstract

Real-time operating systems play an important role in time and safty-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, we implement Earliest Deadline First scheduling algorithm and Stack Resource Policy for time-guaranteed resource sharing.

Acknowledgements

I would like to thank Assistant Professor 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.

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, ().

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, (**?**, pp. 27-31).

μC/OS-III does not implement deadline management but defer this task to the programmer. However, 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

Overview of Micrium μC/OS-III

2.1 Task Model

Current μC/OS-III task model is long-running tasks with statically allocated Task Control Block (TCB) and stacks.

Long-running

Tasks are implemented as normal C functions which do not usually return. Thus, the body of each function normally contain an infinite loop. In addition, tasks are rarely deleted.

Statically allocated TCB and stacks

Due to resource constraints of embedded devices on which μC/OS-III is targeting, dynamic memory allocation is discouraged. Therefore, TCBs along with stacks are usually allocated statically.

Chapter 3

Modifications

3.1 Earliest Deadline First Scheduling

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

3.1.1 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)

3.1.2 Other Notes

wrap around time?

can this fact be exploited? Only one task is running at a time.

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/