Extending uC/OS-III

Arvind Easwaran, Do Nhat Minh

February 17, 2014

Abstract

This is the abstract

Chapter 1

Introduction

Operating systems are collections 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. [1]

... some more text leading to uC/OS-III

uC/OS-III is a priority-based pre-emptive real-time multitasking operating system for embedded systems.

uC/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.

1.1 Task Model

Current uC/OS-III task model is long-running tasks with statically allocated Task Control Block 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 uC/OS-III is run, dynamic memory allocation is discouraged. Therefore, Task Control Blocks along with stacks are usually allocated statically.

Chapter 2

Modifications

2.1 Earliest Deadline First Scheduling

Earliest Deadline First Scheduling is a dynamic scheduling algorithm.

2.1.1 Notes

The EDF scheduler 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)

2.1.2 Other Notes

wrap around time?

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

Bibliography

[1] http://www.cs.utexas.edu/users/witchel/372/lectures/01.OSHistory.pdf