Systems programming for ARM
extending DocetOS

Y1471938

January 10, 2019

Abstract

This document outlines the extensions and modifications made to the DocetOS operating system. The focus of the extensions is on scalability with the aim of allowing the extended docetOS to work with a large number of tasks efficiently. This results in a slight overhead in memory usage but allows for a priority scheduler that can efficiently switch between many tasks, and prevent starvation of individual tasks by ensuring that even the lowest priority task gets occasionally allocated CPU time (scheduling is stochastic, with the task priority determining the probability that the task will get selected). The scheduler incorporates an efficient task sleeping and waiting mechanism which aims to minimise the operations that have to be performed when a task transitions from one state to another. I have provided the user with tools to easily set up inter task communication through the OS channel manager, which is responsible for providing one or more tasks with access to the correct channel, and recycling the channel once the tasks no longer need it. In order to allow the user to easily allocate memory I have created a "memory cluster" that simultaneously aims to prevent memory being wasted and tasks having to wait for memory to become available. The memory provided by the cluster is guaranteed to be 8-byte aligned and can therefore also be used for task stacks. The OS itself uses the memory cluster to allocate the vast majority of its own internal resources such as hash tables, queues, heaps etc. All data structures used by the OS are standalone and available to the user for their own use. They come with convenience functions to quickly create and destroy them e.g "new_hashtable(...)" allocates and initialises a hash table with the desired size.

Contents

1 Coding Standard

2 The Stochastic Scheduler

My plan was to allow DocetOS to support a large number of tasks, which made me come up with the following key requirements for the priority scheduler:

- The scheduler should be pre-emptive, but allow tasks enough time to run to avoid wasting CPU time on frequent context switching.
- CPU time allocated per task should be dependant on the tasks priority, but the highest priority task should not be the only task getting CPU time (not just standard Fixed priority pre-emptive scheduling behaviour where only the highest priority task is selected)
- Task starvation should be avoided, even the lowest priority task should have a non-zero probability of getting selected during task switch.
- Task switching and status changes of tasks should not result in a large overhead of CPU time.