Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

System Construction Course 2018,

Assignment 3

Felix Friedrich, ETH Zürich

Introduction

The memory layout of Minos is absolutely minimal and taylored for this single-core single-threading runtime system. Nevertheless, there must be stack and heap for the running process and the traditional memory layout where the stack top grows towards the heap end provides a good way for balancing between the memory consumption. In this exercise you will learn to use the MMU in order to establish a dynamic between stack and heap memory consumption.

Lessons to Learn

- Getting familiar with the Memory Management Unit of the ARM.
- Understanding IRQ contexts on the ARM and how to deal with page faults.
- Getting hands on experience with heap- versus stack allocation and virtual memory.

Preparation

1. Update your repository
2. Open a console in directory assignments/assignment3

1 Virtual Memory

Currently, the Minos memory layout is static: stack and heap sizes are fix. For known applications, this is not an issue as the sizes can be set in the kernel. However, to support a range of unknown applications that might have different requirements on the stack and heap sizes, a more flexible way of allocating memory to the stack and the heap is required.

Again, in order to give some hints there are comments starting with `STUDENT` in the source code that guide you where you need to amend or modify the source code.

1. You will have to build the Minos Kernel whenever you change the memory Management in module `Kernel`, because it is statically linked to the Minos kernel. As in the previous exercise, you build the kernel using the commands in MakeMinos.txt, e.g. by calling

   ```
   oberon execute MakeMinos.txt
   ```

2. Get familiar with the Memory Management Unit of the ARM.

   Currently, the system starts up with a 1:1 memory map. See procedure `IdentityMapMemory` in module Minos/RPI.Platform.Mos

   Modify the InitMMU procedure in module Minos/RPI.Kernel.Mos such that only one page is allocated each for the stack and the heap during boot up. We use only first level page table entries (1 MByte).

3. Consult the ARM v7 Architecture Reference Manual and / or the Cortex A7 Technical Reference Manual in order to find out how to identify the page that was accessed when observing a page fault trap.

4. If the stack or the heap exceed their current limit, a data abort trap (page fault) should occur. Check this by allocating excessive heap and/or writing recursive procedures. Test commands `TestAllocation.Heap <n>` and `TestAllocation.Stack <n>` are provided with module TestAllocation.Mos.

   Add to the data abort trap handler `DataAbort` in module `Kernel.Mos` a functionality to map one spare page either to the stack or the heap depending on the trap source.

5. **Optional:** Try to come up with a solution with a minimal number of data abort exceptions: refine your implementation making use of the fact that heap allocation happens explicitly while stack allocation happens implicitly. Requires a modification of module Minos/Heaps.Mos

Documents

- ARM Architecture ReferenceManual ARMv7-A and ARMv7-R edition in the `documents/rpi` folder of the repository
- Cortex-A7 MPCore Technical Reference Manual in `documents/rpi` folder of the repository
- System Construction Lecture 3 slides from the course-homepage `http://lec.inf.ethz.ch/syscon`