CSE4354/5354 Real-time Operating Systems
Fall 2024 Mini Project (Memory Protection Unit and Memory Manager)

1 Overview

The goal of this assignment is to introduce split privileged/unprivileged operation and provide memory management and memory protection that will be used in the RTOS project.

2 Hardware Description

Microcontroller:
An ARM M4F core (TM4C123GH6PMI microcontroller) is required.

Serial interface:
If using the EK-TM4C123GXL evaluation board, then the UART0 tx/rx pair is routed to the ICDI that provides a virtual COM port through a USB endpoint.

3.3V supply:
The circuit is powered completely from the 3.3V regulator output on the evaluation board.

3 Software Requirements

A virtual COM port using a 115200 baud, 8N1 protocol with no hardware handshaking is used to communicate with this project.

For this design, you must set the ASP bit in the CONTROL register so that the thread code uses the PSP (the handler mode will always use the MSP). Make sure to place an instruction barrier instruction (ISB) after the write to the CONTROL register so that instructions use the correct stack.

The software must provide the following functions. As in the shell interface, you cannot call C library functions.

If a bus fault ISR occurs, display “Bus fault in process N”, where N will be a variable provided by the OS. Or now, just use a variable named pid.

If a usage fault ISR occurs, display “Usage fault in process N”, where N will be a variable provided by the OS. Or now, just use a variable named pid.

If a hard fault ISR occurs, display “Hard fault in process N”, where N will be a variable provided by the OS. Or now, just use a variable named pid. Also, provide the value of the PSP, MSP, and mfault flags (in hex). Also, print the offending instruction and data addresses. Display the process stack dump (xPSR, PC, LR, R0-3, R12.

If an MPU fault ISR occurs, display “MPU fault in process N”, where N will be a variable provided by the OS. Or now, just use a variable named pid. Also, provide the value of the PSP, MSP, and mfault flags (in hex). Also, print the offending instruction and data addresses. Display the process stack dump (xPSR, PC, LR, R0-3, R12. Clear the MPU fault pending bit and trigger a pendsv ISR call.

If a pendsv ISR occurs, display “Pendsv in process N”. If the MPU DERR or IERR bits are set, clear them and display the message “called from MPU”.

Understand how all of these ISRs are called by code. Using the circuit for the main project detailed in class, assign each of the push buttons to invoke intentional errors to trigger the ISRs above.

You will also need to create a simple memory manager than allocates memory from the global heap. The function prototype is

void * malloc_from_heap (int size_in_bytes)

The name is chosen to not conflict with the malloc function in C which is not implemented for our OS.

The memory allocated will be rounded up to the nearest multiple of 512B if the allocation requested is <= 512B and rounded up to the nearest multiple of 1024B otherwise. The memory address of all allocated memory chunks will be aligned to a multiple of 512 for 512B allocations and a multiple of 1024 for N*1024B allocations. The mix of 1024B and 512B allocations is a design decision. CSE 4354 will also support 1536B allocations on the boundary between the 512B and 1024B subregions.

Implement free() function as well.

Study in detail in the ARM documentation the relative prioritization of the MPU regions before starting the following steps:

To configure the MPU regions, follow these steps:

Overall Access (optional background rule)

To make the following steps easier to debug, add an all memory background rule that allows RW access to all memory for both privileged and unprivileged code. You can remove access for the unprivileged code later or leave it in the code (see the note in the peripheral access section on this).

Flash Access

Implement a function, void allowFlashAccess(void), that creates a full-access MPU aperture for flash with RWX access for both privileged and unprivileged access.

Call allowFlashAccess(). While in privileged mode, verify the program still runs as expected when turning on MPU with flash access control.

Temporarily switch to unprivileged mode and verify that you can access flash and run code.

Revert to privileged mode before continuing.

Peripheral Access

Implement a function, void allowPeripheralAccess(void), that creates a full-access MPU aperture to peripherals and peripheral bitbanded addresses with RW access for privileged and unprivileged access. If you choose to leave the all access rule, then this is not needed.

Call allowFlashAccess() and allowPeripheralAccess(). While in privileged mode, verify that the program still runs as expected when turning on MPU with flash, RAM and peripheral access control.

Temporarily switch to unprivileged mode and verify that you can access flash (run code) and r/w memory.

Revert to privileged mode before continuing.

SRAM Access (All)

Create a function, void setupSramAccess(void), that creates multiple MPUs regions to cover the 32KiB SRAM (each MPU region covers 8KiB or 4 KiB, with 8 subregions of 1KiB or 512B each with RW access for privileged mode and no access for unprivileged mode. It may be useful to disable the subregions to start.

Create a function uint64_t srdBitMask = createNoSramAccessMask(void) returns the values of the srdBits to allow no access to SRAM.

Create a function applySramAccessMask(uint64_t srdBitMask) that applies the SRD bits to the MPU regions. This function will only be called in privileged mode.

Create a function void addSramAccessWindow(uint64_t *srdBitMask, uint32_t *baseAdd, uint32_t size_in_bytes) that adds access to the requested SRAM address range.
Call allowFlashAccess(), allowPeripheralAccess(), setupSramAccess(), createNoSramAccess(), and setSramAccessWindow((unit32_t *)0x20000000, 32768).

While in privileged mode, verify that the program still runs as expected when turning on MPU with flash, RAM and peripheral access control.

Temporarily switch to unprivileged mode and verify that you can access flash (run code) and r/w memory.

Revert to privileged mode before continuing.

SRAM Access (Restricted)

Call uint32_t* p = malloc_from_heap(1024)

Call allowFlashAccess(), allowPeripheralAccess(), setupSramAccess(), createNoSramAccess(), and setSramAccessWindow(p, 1024) to create access to the newly allocated memory.

Dereference the pointer and write to the address (*p = value).

While in privileged mode, verify you can still access ram in the allocated range of SRAM.

Switch to unprivileged mode and verify that you can access flash (run code) and r/w memory from the peripherals.

While in unprivileged mode, verify that accessing SRAM outside of the allocated range is not allowed (you will see an MPU ISR with information about the errant read/write from memory).

4 Testing

Your code will be tested in the ERB 125 lab by the grader.

5 Deadlines

The assignment is due on the date and at the time indicated in the syllabus. This is an individual project, so do not exchange or share code with others.

6 Safety Issues

When utilizing the lab, please observe all safety rules as stated in the syllabus.

Have fun!