CS 452 Kernel 3

Felix Fung (f2fung)
Dusan Zelembaba (dzelemba)

June 4, 2013

1 How To Run

```
> load -b 0x00218000 -h 10.15.167.4 "_________________"
> go
```

2 Submitted Files

Files listed here can be found under `_________________`

2.1 md5sums

2.2 Header Files

`context_switch.h` - Function definitions for compiler to use our assembly functions.
`debug.h` - Macros wrapping debugging functions that can compile away.
`messenger.h` - Function definitions for messenger.
`nameserver.h` - Function definitions for starting the nameserver.
`queue.h` - Function definitions for our queue implementation.
`rps_server.h` - Functions definitions for starting and using the rock, paper, scissors server.
`scheduler.h` - Function definitions for scheduler.
`string.h` - Functions definitions for string library.
`stdlib.h` - Function definitions for commonly used functions.
`syscall.h` - Function definitions for syscalls.
`task.h` - Contains Task structure and function definitions.
`timer.h` - Function definitions for timer.

2.3 Source Files

`context_switch.s` - ARM assembly used to switch in and out of tasks and kernel mode.
`debug.c` - Debugging facilities.
`kernel.c` - The infinite kernel loop. Interrupts are processed here and tasks and scheduled.
`main.c` - Beginning of execution, described below.
`messenger.c` - Coordinates message passing.
`Makefile` - Our makefile.
`nameserver.c` - Implementation of the nameserver, and WhoIs/RegisterAs syscalls.
`queue.c` - A simple queue implementation.
`rps_server.c` - Implementation of the rock, paper, scissors server and wrappers for Send that clients can use to talk to the rps server..
`scheduler.c` - Our scheduler.
`stdlib.c` - Common functions. Not actually a standard library.
`strings.c` - String library.
`syscall.c` - System calls.
`task.c` - Creating and managing tasks.
`timer.c` - Timing functions.

2.4 Test Files

`tests/basic_test.c` - This is a basic test. (This is only run for debugging)
`tests/message_passing_test.c` - Tests all aspects of Send, Receive, Relpy.
`tests/multiple_priorities_test.c` - Tests the correct scheduling of multiple tasks with different priorities.
`tests/nameserver_test.c` - Tests WhoIs & RegisterAs.
`tests/rps_server_test.c` - Implements a set of clients for the nameserver.
`tests/srr_speed_test.c` - The test that is run to analyze our performance.
`tests/syscall_speed_test.c` - Test the time required to execute various sys calls.
`tests/task_creation_errors_test.c` - Tests error return values for task creation.

`run_tests.h` - Header for run_tests.c.
`run_tests.c` - Calls all our tests.
`test_helpers.h` - Header for test_helpers.c. Different types of asserts and such.
`test_helpers.c` - Test helpers. Asserts and other things.

3 Program Description

3.1 Scheduler

Our old scheduler used an array of queues to implement the priority queues. It was mentioned that this approach would be too slow once we added more priorities, so we upgraded our implementation to use a bitmask.

Our bitmask has one bit for every priority. If a bit is set, that means there are tasks's ready to be run for that priority. So, when a queue empties we set its bit to 0 and when a queue gets a first element we set its bit to 1. Then `scheduler_get_next_task` becomes a find lowest bit operation (0 is highest priority). To do this we used the linux kernel's implementation of fls.

3.2 Context Switch

To support hardware interrupts we had to enhance out context switch. Our old context switch stored registers r4 - r13, the pc of the user, and the CPSR of the user on the user's stack. For hardware interrupts, we also had to store registers r0 - r3. To do this, we created a wrapper around `k_enter` and `k_exit`: `hwi_enter`.

The wrapper saves registers r0 - r3, the user's CPSR and PC on the user's stack. Then it creates a `Request` object with the interrupt type and sets up the SPSR_pc and LR_pc so that `k_exit` will jump back into the wrapper. It then jumps to `k_enter`.

Once the kernel finishes, we get back into the wrapper and restors the user's scratch registers, CPSR and PC.

3.3 AwaitEvent

3.4 Clock Server

4 Program Output