CS 452 Assignment 1

Felix Fung (f2fung)
Dusan Zelembaba (dzelemba)

May 24, 2013

1 How To Run

```
> load -b 0x00218000 -h 10.15.167.4 "ARM/f2fung/main.elf"
> go
```

2 Submitted Files

Files listed here can be found under `/u0/f2fung/cs452/a1`

2.1 md5sums

```
2fd65abca13a814332e40d8b5fed938c  all_tests.h
db8bae8ff95a9ccc357a4e26ad12dbab  assignment_1_test.c
4b6eaafaafb3d99259a3faaa4ba640ab  basic_test.c
b8f6809658948d4086826b1e3c0241c3  context_switch.h
cef010d3ef08df27f78a2298e86e9a81  context_switch.s
ee14573108ac7fb308e3924a48e96bba  context_switch_speed_test.c
00c982f510dc973dc60313ebddb8a03d  kernel.c
664366bb7fe0a090ebdfe1b3009a14ba  kernel.h
f8775577c797a37a4d3f5e7874dff684  main.c
282f4ca1b5d6c9e1168db6528490ea30  Makefile
2769176d92c2eb972f29957500c870c8  multiple_priorities_test.c
454ff6f490bbe05287f63c619a588427  queue.c
987fb24e1e06f8fe08ecb13be58702a2  queue.h
1f272b215198236dcc88619f44b20438  run_tests.c
0e390e153530b05118c87063a5dd3120  run_tests.h
bfb8b08941df42986674cab9835630a3  scheduler.c
bc03c446589619c1249da7313e37f850  scheduler.h
c3a1f01398fffc6868e1bad847f30dab  syscall.c
9c20435c4901586afe2c7bc8a06dc15f  syscall.h
d61d08108e30320e183c050624160bf6  task.c
11018b286cf38a3535c017d62d3ceb4e  task_creation_errors_test.c
17c617857a7f133ca6da9d62a52bdc7c  task.h
```

```
75dec88a5d7b4ab2cb4805b1f196faf2  test_helpers.c
c6eb413e3b7e544e8051e783175595c2  test_helpers.h
```

2.2 Header Files

`context_switch.h` - Function definitions for compiler to use our assembly functions.
`queue.h` - Header for our queue implementation.
`scheduler.h` - Function definitions for scheduler.
`syscall.h` - Function definitions for syscalls.
`task.h` - Contains Task structure and function definitions.

2.3 Source Files

`context_switch.s` - ARM assembly used to switch in and out of tasks and kernel mode.
`kernel.c` - The infinite kernel loop. Interrupts are processed here and tasks and scheduled.
`main.c` - Beginning of execution, described below.
`queue.c` - A simple queue implementation.
`Makefile` - Our makefile.
`scheduler.c` - Our scheduler.
`syscall.c` - System calls.
`task.c` - Creating and managing tasks.

2.4 Test Files

`basic_test.c` - This is a basic test. (This is only run for debugging)
`context_switch_speed_test.c` - This tests the time it takes to Pass(), reschedule, and come back. (This is only run for debugging)
`multiple_priorities_test.c` - Tests the correct scheduling of multiple tasks with different priorities.
`run_tests.h` - Header for run_tests.c.
`run_tests.c` - Calls all our tests.
`task_creation_errors_test.c` - Tests error return values for task creation.
`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 Main

This is where the kernel's execution begins. We first set a known address ($0x28$) to point to our interrupt handler. Because Create() can only be called in user-mode (causing a system interrupt), we manually call kernel methods to create

the first user-task and stick it into the scheduler in main. For the purposes of the assignment, this is done in assignment_1_test.c.

3.2 Context Switch

We save all the registers on the top of the user's stack. During task create, we fill the user's stack with initial register values so that the first context switch into the user task behaves exactly like other context switches.

3.3 Data Structures

3.3.1 Task Id Provisioning

We use a queue which is initialized with the universe of available task ids (0-1023). These task ids point into an array of Task structures. We recycle task ids by pushing returned task ids into the back of our queue.

3.3.2 Task Descriptors

Task descriptors are structs. They contain their tid, parent's tid, priority, return value for their last system call, stack pointer and a pointer to their stack space. This is suboptimal, but it is thus far unnecessary to optimize.

3.3.3 Scheduler

We use 4 queues, one for each priority we define. Queues are implemented by circular arrays.