CS 452 Assignment 2

Felix Fung (f2fung)
Dusan Zelembaba (dzelemba)

May 31, 2013

1 How To Run

```
TODO: Compile this and put it somewhere
> load -b 0x00218000 -h 10.15.167.4 "ARM/f2fung/THISISBAD.elf"
> go
```

2 Submitted Files

Files listed here can be found under `/u0/f2fung/cs452/a2`

2.1 md5sums

```
TODO: Run md5sum*
```

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` - TODO.
`queue.h` - Function definitions for our queue implementation.
`rps_server.h` - TODO.
`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` - TODO.
`queue.c` - A simple queue implementation.
`rps_server.c` - TODO.
`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` - TODO.
`tests/multiple_priorities_test.c` - Tests the correct scheduling of multiple tasks with different priorities.
`tests/nameserver_test.c` - TODO.
`tests/rps_server_test.c` - TODO.
`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 Message Passing

To make better use of the cache, we've made a space-saving statically allocated data structure for message passing. We create a fixed-size array of INBOXes,

MAX_TASKS in number. Each INBOX contains fields we reuse to contain EITHER a sent message and reply address (params of Send()), or the receiving address (params of Receive()). There are no collisions because Send() and Receive() cannot be concurrent for a single task. Each INBOX contains a NEXT pointer, which may point to another INBOX or NULL allowing INBOXes to be chained together in a linked-list fashion. Then for MAX_TASKS, we also store 'head' and 'tail' pointers to INBOXes to represent the receive queue for each task in constant space relative to the length of the receive queue.

In summary, when blocking, tasks will write their addresses to their own INBOX, and it'll be chained into the receiving task's linked-list. By reusing INBOXes for future messages, we never allocate more memory.

3.2 Nameserver

3.3 Rock, Paper, Scissors Server

3.4 Timing

All timing code is being run in user-mode. When testing the send-receive-reply speeds, we only measure the full round-trip. In the case of send before receive, we begin timing before being send blocked to when the task is active again. (Implying that receive and reply have been called from the other task). We ensure there are only two tasks and so our sender task goes directly from ready to active without further delay. In the case of receive before send, we begin timing before being receive blocked and after we return from reply. Based on our implementation, these times all contain the time required to copy messages across buffers.

Our memcpy implementation is naive and I believe a lot of our time is being spent here. Increasing message size adds a significant performance hit as seen in our performance results.