CS452 Kernel 2

Christoph Ulshoefer (20751216) and Thomas Broadley (20522223)

January 29, 2018

1 Running the kernel

Execute the following commands in RedBoot to start the kernel:

```
load -b 0x00218000 -h 10.15.167.5 "ARM/csulshoe/k2.elf"
go
```

Our K2 ELF file is stored at `/u3/cs452/tftp/ARM/csulshoe/k2.elf` in the CS Student Computing Environment. Our K2 source code can be found at https://git.uwaterloo.ca/csulshoe/cs452/tree/k2-done.

2 Implementation

This section describes the three major additions to our project that are part of K2: message-passing, the nameserver, and the RPS user tasks.

2.a Message-passing

In K2, we added three new task states: receive-blocked, send-blocked, and reply-blocked. We also implemented three syscall wrappers to facilitate message-passing: `Send`, `Receive`, and `Reply`. Finally, we added a send queue struct that uses the same data structure as the ready queues: a doubly-linked circular list.

We wrote generic implementations of the queue methods so that we could use the same implementations for both queue types. After defining the macros `QUEUE_TYPE`, `QUEUE_NEXT`, and `QUEUE_PREV`, `kernel/src/multitasking/ready_queue.c` and `kernel/src/multitasking/send_queue.c` both import `kernel/src/queue.h`. These macros specify the type and previous/next element accessors to be used by generic queue method implementations contained in the latter file. Even though we have only written unit tests for one of the concrete implementations, we can be sure that the other implementation would pass the same tests.

Like the ready queues, the send queues operate on a FIFO basis. This means that a high-priority task could be stuck on a send queue while the receiver is processing a message from a low-priority task. However, if we implemented send queues as a priority queue based on task priority, low-priority tasks' messages might never be received. We consider the second issue to be worse than the first. If it turns out that round-robin is not effective, we are prepared to change our send queue implementation; we will likely use Dijkstra's algorithm in an later assignment, so we will have to implement a priority queue eventually.

We implemented `Send`, `Receive`, `Reply` and according to the kernel specification, which states that messages should be of type `char *`. Since the message types we used for K2 require little formatting or parsing, we found using strings as messages worked fine. Given that we will likely use more complex message formats in the future, we intend to change the message type to `void *`, to make it easier to use arbitrary structs as messages.

2.b Nameserver

The nameserver accepts two types of messages: `RegisterAs` messages, which start with an 'R; and `WhoIs` messages, which start with a 'W'. The latter part of both kinds of messages contains the non-empty null-terminated name to register as or to look up.

The nameserver maintains an array of task names and an array of task IDs. The name at a given index in the former corresponds to the task ID at the same index in the latter.

To respond to a `RegisterAs` call, the nameserver first searches for the given name in the array of tasks. If it is found, it replaces the corresponding task ID with the ID of the task that called `RegisterAs`. Otherwise, it adds the name-ID pair to the end of the arrays.

To respond to a `WhoIs` call, the nameserver searches for the given name in the array of tasks. If the name is not found, the nameserver returns an error.

We believe that we will create most user tasks near the start of our kernel's execution, and that most user tasks will only query the nameserver a few times at the start of their execution. Therefore, we do not consider making `RegisterAs` and `WhoIs` more efficient a priority. If our assumptions are incorrect, we could store names using a trie, a radix tree, or a similar data structure.

2.c RPS user tasks

We wrote three tasks for the K2 RPS test: the first user task, the server, and the client.

2.c.1 First user task

The first user task creates the nameserver, then the RPS server, then each RPS client. By creating the nameserver first, we ensure that `Send` calls to the nameserver made by the RPS clients will not fail.

The first user task is given priority 10, a higher priority than any other task. This is so that it can create all of the other tasks at once and then exit, allowing the other tasks to interact.

2.c.2 Server

The server registers with the nameserver under the name "RPS". It accepts three types of messages, all one character long:

1. Sign-up ('B'), which replies with 'N' if the task already has a partner or 'Y' once it has been matched with a partner;
2. Play ('R', 'P', or 'S'), which replies with:
    - 'N' if the task doesn't have a partner or if either the task's or its partner's throw is invalid,
    - 'W' if the task has won,

- 'L' if the task has lost, or
- 'D' if the game was a draw; and

3. Quit ('Q'), which replies with 'N' if the task doesn't have a partner or 'Y' otherwise.

If a client is sent an error message because it tried to play or quit while not having a partner, that task is allowed to sign up again.

On sign-up, the server adds tasks to a task ID buffer of size two. Once the buffer is full, it matches the two tasks in the buffer and clears it.

To keep track of matched tasks, the server uses an array of 64 task IDs named `partners`. When it matches two tasks, it places each task ID in the location in the array corresponding to its partner's task ID. For example, if the server matched tasks 5 and 10, it would store 5 in `partners[10]` and 10 in `partners[5]`. This allows constant-time lookup of each task's partner.

The nameserver and the RPS server both have priority 5, lower than the first user task but higher than the clients. Both of these servers will spend most of their time blocking on a `Receive` call. When they receive a message, we want them to process the message as soon as possible so that they can quickly unblock the sender.

2.c.3 Client

Each client discovers the task ID of the nameserver with a `WhoIs` request, signs up with the server, plays a number of games decided by its task ID, and quits. If the client's task ID is even, it signs back up and plays more games before quitting again.

Each client has access to a shared linear congruential pseudorandom number generator (pRNG). The clients use this pRNG to choose throws for each game. Since the pRNG's seed is constant across runs, the output of the RPS tasks is also constant. This makes it much easier to test that the output from K2 is correct. Instead of writing a test that parses the output to find errors, we can simply check the correctness of the output once and assert that it does not change on later runs.

3 Extra features

We implemented an abort and undefined instruction handler as part of K2. We also wrote end-to-end tests for our kernel using Python and the QEMU emulator.

Finally, we added continuous integration to our development process. Every time we create a PR, we build our kernel for the ARM box and for the Versatile/PB (our emulated system-on-a-chip), then run our unit and E2E tests. We only merge PRs if all of these checks are successful.

3.a Abort and undefined instruction handler

We couldn't extract useful information from the output that RedBoot provides when a segfault occurs. We decided to write our own handler and modify low memory so that this handler is invoked if a data abort, prefetch abort, or undefined instruction occurs. The handler prints the exception type, the current processor mode, and the instruction that caused the exception, then exits cleanly to RedBoot.

3.b End-to-end tests

Each E2E test is a user task that executes a set of syscalls and prints output to the terminal. We wrote a Python script that connects to the QEMU emulator over TCP, runs the appropriate user task, and retrieves this terminal output. It then checks that actual output against an expected value.

To run the appropriate test, the script passes the test name over TCP to the emulator. The name is received by a test runner user task, which checks the name against its list of tests and runs the appropriate one.

4 RPS output

This section contains an annotated (and occasionally abbreviated) transcript of the output of our K2 RPS test tasks.

```
Task 4 signed up to play
Task 5 signed up to play
Matched tasks 4 and 5
Task 6 signed up to play
Task 7 signed up to play
Matched tasks 6 and 7
```

The first client has task ID 4 because the first user task has ID 1, the nameserver 2, and the RPS server 3.

```
Task 4 played P
Task 5 played R
Task 4 wins with P, task 5 loses with R
Task 6 played S
Task 7 played S
Tasks 7 and 6 draw with S
Task 5 played R
Task 4 played S
Task 5 wins with R, task 4 loses with S
Task 7 played P
Task 6 played S
Task 6 wins with S, task 7 loses with P
[removed 12 lines]
```

Each pair of tasks plays four times. The pairs swap games because of the FIFO nature of the RPS server's send queue. When a pair of tasks finishes a game, they immediately submit two new moves and put themselves back on the RPS server's send queue. The pair of tasks will be behind the tasks in the other pair, since those tasks were already on the queue, waiting for the server to service the first pair of tasks.

Within each pair, the task that plays first swaps each game. This is because, when a game finishes, the server first replies to the message that sent its throw second.

```
Task 4 played R
Task 5 quit
Task 6 played P
Task 7 played S
Task 7 wins with S, task 6 loses with P
```

Task 5 quits after four games. Task 4, having already made a move, is informed immediately that its partner has quit. Tasks 6 and 7 will play one more game.

```
Task 7 played P
Task 6 played P
Tasks 6 and 7 draw with P
Task 5 exiting
Task 4 signed up to play
Task 6 played S
Task 7 quit
Task 7 exiting
Task 6 signed up to play
Matched tasks 4 and 6
```

Since task 4 has an event task ID, it signs up to play again. When task 7 quits, task 6 also signs up again and the server pairs the two tasks.

```
Task 4 played S
Task 6 played S
Tasks 6 and 4 draw with S
[removed 12 lines]
Task 6 played P
Task 4 quit
Task 4 exiting
Task 6 exiting
```

Tasks 4 and 6 play five times, until task 4 quits and exits. Task 6 also exits because it is informed immediately that task 4 has quit. All user tasks have exited, so the kernel exits cleanly to RedBoot.

5 Benchmarks

Please consult `k2benchmark.txt` in our repository. We apologize that the layout may look weird in your text editor because of our long group name (`'); DROP TABLE trains; --`).

We also did more granular benchmarking, measuring how long each task spent in the context switch, in the scheduler, and copying messages between tasks. We found that the scheduler accounted for between 50% and 70% of the time. Copying the message took 5% of the total time for the 4-byte message and 15% to 25% for the 64-byte message. Kernel entry and exit took between 5% and 15%.

6 File and repository hashes

The commit before adding this report had SHA `95940fbc745dadb184d7da0edf217517978e7f48`.

6.a MD5 hashes

```
3baa32ac2b0d0342c6975830905a3da0  ./.gitignore
4bbdec35a4b9dc9ca8615b1e321de502  ./.gitmodules
b5698bed8e2a55c5ef2028266b8f8973  ./.travis.yml
82bceca3784a9a6688bbc1438fa72eea  ./Doxyfile
d7a19a053f2c7a4d0bde5ec268885f3a  ./Makefile
61e581be7fe48482d7c4ecbecf8ea808  ./README.md
9c611ad7c1eacdd54944c946b2363c37  ./include/myio.c
0cc48458ae2a51df2085574baae1a15b  ./include/myio.h
1d09486aa883070ac1a06c2bcbdfb184  ./include/mytimer.c
7d92a29cc1190f5c6a8c0ace259b0ad3  ./include/mytimer.h
5e27447bd4c3d0de2c4ef58654a6b59e  ./include/timer_data.h
74c63a11fc85af6c146dbcb5fa19b14a  ./k2benchmark.txt
eafdaf77048f8e5a5e228a8cb2fc9ea2  ./kernel/asm/kernToUser.s
2b40643c713897d963f460ac0e87f155  ./kernel/asm/startup.s
5752364606cb8e34250fefc41a3fd353  ./kernel/asm/trap.s
5900a531da2411921c91efa2d19e123e  ./kernel/include/labenv/timer.c
25e2089c22afbd139656129c4cd272f0  ./kernel/include/labenv/timer.h
867d06b95f132fbd63d2ba63cae7ef21  ./kernel/include/labenv/ts7200.h
ca1ae0528fde83fd2013a5bd75085b70  ./kernel/include/versatilepb/timer.c
f8a5328030819cd012d98cf4728021c2  ./kernel/include/versatilepb/timer.h
a52772c927a3f3937e2af9f2536c5433  ./kernel/include/versatilepb/versatilepb.h
48f88c34f1e256029ff2e41ac7e00cc8  ./kernel/src/cp_vec.c
503fb56ff1929bcc8d7a6b3d19b2524e  ./kernel/src/cp_vec.h
fec1b3333a526018a7878b2c2777df6c  ./kernel/src/handle_abort.c
7775959970465ee980955b1e98016b04  ./kernel/src/interrupt.c
8afce47cf0fbc6ed282b68c6be84e72b  ./kernel/src/interrupt.h
034be81e5bcaacbbd751e1f50c98999c  ./kernel/src/kassert.c
d8338a43bb4b734d6d3c1567378164f4  ./kernel/src/kassert.h
626eb8ce546c4388ed5104575e07c57c  ./kernel/src/multitasking/messaging.c
fd5db1d578749c314fe742ee2a2acae6  ./kernel/src/multitasking/messaging.h
db3c340adf6f2f90834328e4a7414daf  ./kernel/src/multitasking/ready_queue.c
8a7c04896ef89aa935a7092691112c0e  ./kernel/src/multitasking/ready_queue.h
b0e53853dfce269b5786549f25134511  ./kernel/src/multitasking/schedule.c
51a0cd22809469a207eb31fd01c4802f  ./kernel/src/multitasking/schedule.h
4412b55a51c1db2b4438f980342597f2  ./kernel/src/multitasking/scheduler.c
e765318e5fc805bb319cd98dbe2d1d1f  ./kernel/src/multitasking/scheduler.h
ce79a33fb4e09a3b869b2d2f187df6a7  ./kernel/src/multitasking/send_queue.c
d2fd55f45ab57e8e5c75a5a072bb74ec  ./kernel/src/multitasking/send_queue.h
a0c9caddb2a9024cf68f0762693d3865  ./kernel/src/multitasking/task.c
924b89ade495cdc98709f5624b950dcb  ./kernel/src/multitasking/task.h
ad867280e3d7d5a92d9a094cfa3ce330  ./kernel/src/queue.h
941801b8944a6a5da1365e95816d1e1e  ./kernel/src/syscall/syscall.c
6f94294551a7fd1f454ef673ecefb6ff  ./kernel/src/syscall/syscall.h
bec5edf5b40af4209c289f8c83f248ef  ./lib/project/attributes.h
e424cefdac5a6beeaddb636aea85919c  ./lib/project/benchmark.c
3b048b3ec9b6fee9a4ff5f054de90a14  ./lib/project/benchmark.h
c392cc152aa5cf68d6ff2a481718d4b1  ./lib/project/buffer.c
6a206871ca8335c2eb17dc27287195a0  ./lib/project/buffer.h
0dcbabb045b2b3c594e84163645d7bb9  ./lib/project/codes.c
0fb962e753e573331b6b62989632d2b3  ./lib/project/codes.h
38424f2856725ba4ce4919eefe5e3142  ./lib/project/crash.h
608c3732de6719e206bf1c76772bd549  ./lib/project/crash.s
ca20845dbe0c6433f03d3fc0219021c6  ./lib/standard/stdlib.c
49c7f159768a8c87f00dedc7735867d5  ./lib/standard/stdlib.h
439a27b05d2f02e1065a560481167b45  ./main.c
04a99fa4e57230b2b2b7a525a1105097  ./main.ld
```

```
d9cd792c1413ac79bf45f799ae903f42  ./reports/a0-csulshoe.pdf
2d81bda8e6453ea87e964edefb8d37c5  ./reports/k1.pdf
1c7a429e1ced54dbb23098d3078a30e8  ./stats.md
da4d2d5a9d3d7971c83928f051ef6b58  ./test-resources/assert.c
3b0613fd033e3b10442f69ef7cd84119  ./test-resources/assert.h
5b00bd5131939d80c1244cc56d51867a  ./test/Makefile
d41d8cd98f00b204e9800998ecf8427e  ./test/e2e/__init__.py
b6e09723bdadead9fb867093f90c183a  ./test/e2e/qemu_tcp_wrapper.py
f1e50c9027d900c43570595e642fd518  ./test/e2e/snapshots/k2.txt
715aa6b5953a0e59e10e463fc680971a  ./test/e2e/test_kernel_demo.py
c80009eeb7b6b7f1b858d71f53cc5133  ./test/e2e/test_messaging.py
50f189623bb32bb07f840e9e948ecdea  ./test/e2e/test_nameserver.py
0eaf78640da1c90c71b02f7f499d42b9  ./test/e2e/test_segfaults.py
bc14b754a90bd8d836cfdb4790726779  ./test/e2e/test_test.py
3834f5e022525034f78b19b6d5b764fa  ./test/unit/test_all.cc
fef09e15aad9df144989ba98bdd2abe1  ./test/unit/test_all.h
5987f0d06178b912a7972264b9340349  ./test/unit/test_buffer.cc
d12680bd57276187b1e7e344ac5a71b7  ./test/unit/test_buffer.h
070019a71d706c33119827ecb85cc1a7  ./test/unit/test_messaging.cc
46ec33cd074fbbbca13543e5787d9b9a  ./test/unit/test_messaging.h
26b0a0b5953b3f0dbc4c04ee0ff5636d  ./test/unit/test_ready_queue.cc
917d4146f8727af1b9986cd4b94565a3  ./test/unit/test_ready_queue.h
cbc64f713a75291b99a29b90a35340b4  ./test/unit/test_scheduler.cc
c31f7630705c36d377356eace6e86c9b  ./test/unit/test_scheduler.h
e10279b12bdd6bbdb15dca5a98f04f12  ./test/unit/test_task.cc
36889e34775aefba76462f7ede00349b  ./test/unit/test_task.h
50bfc80306b3323a944d7ff54d976bf2  ./usr/a0.c
cb3a97bfb7a6d80cb876c69e67930bf6  ./usr/a0.h
f203d61b324bd43535ae6b4f294fd95a  ./usr/a0/a0terminal.c
637cf17011f4e4e14c1835b55a4bef4c  ./usr/a0/a0terminal.h
2f59b880230fd85f2ec9293ddb3f2453  ./usr/a0/a0trackstate.c
c858503929343d5c24870a5f4dff90af  ./usr/a0/a0trackstate.h
09be367c99cfb179c3552f5d2fd6b6d7  ./usr/k1.c
758b67d7c795e165ba195c0d39d6bbe0  ./usr/k1.h
95cb970b7daeb50ea79a134b4aed2ba5  ./usr/k2.c
5f1339578b940291394306bb2e4f8ac9  ./usr/k2.h
47218452282dc89e73e6e268ac405c1a  ./usr/nameserver.c
9c46f1f45256a7bd247b67708ab21cce  ./usr/nameserver.h
3a5cb2cb079d41415f4100d6cfd956b1  ./usr/test/messaging/test_messaging.h
539ef725e22cc47d6d91ce0fc4ee0c8e  ./usr/test/messaging/test_messaging_basic.c
95cdf5812efd4b9a7c27e83c2808822a  ./usr/test/messaging/test_messaging_exit_with_blocked.c
225afd18f178ebad1e71f7608926b2ed  ./usr/test/messaging/test_messaging_fifo_send.c
f0abb91510a1c1433da5977ba2a1881d  ./usr/test/messaging/test_messaging_invalid_tid.c
a100c3af0433f9944c4cf86b269043a2  ./usr/test/messaging/test_messaging_receive_before_send.c
eb2a6f59b99f160c729403b14b21590a  ./usr/test/messaging/test_messaging_reply_target_zombie.c
522cba488790f761d2ee9a28ceeafa14  ./usr/test/messaging/test_messaging_same_priority.c
b9809a24eeef4d6fb6dd4723adc9e6fd  ./usr/test/messaging/test_messaging_send_recipient_zombie.c
2f5b23f671bcf4e6253a68fedcd8dfc2  ./usr/test/messaging/test_messaging_sequence.c
fc0283ddb6231a75d6cd9a86e4b653d9  ./usr/test/messaging/test_messaging_tree.c
d8ef7ac44343b493b5a900b0a2d2c233  ./usr/test/messaging/test_messaging_truncation.c
29fedcb6f5835332dde0f1f21ee1e066  ./usr/test/nameserver/fake_nameserver_functions.c
29b393264766e7785e412f42c0fc274b  ./usr/test/nameserver/fake_nameserver_functions.h
aa44a48fd8d751e6150561e6182fa37e  ./usr/test/nameserver/test_nameserver_errors.c
49fb635d336c9088b11267f6d5b4c585  ./usr/test/nameserver/test_nameserver_errors.h
60aab909b89397c55ac53a55beff87ab  ./usr/test/nameserver/test_nameserver_happypath.c
a611abae3660fd3dad599179cc78a10c  ./usr/test/nameserver/test_nameserver_happypath.h
d1187f6bb659866b1818bea0b9c06d9d  ./usr/test/nameserver/test_nameserver_too_many.c
74207e20a985228b03069084c3939371  ./usr/test/nameserver/test_nameserver_too_many.h
```

```
c516c2b49ee82386db3ce7bfcdd1fb93  ./usr/test/nameserver/test_nameserver_wrapper_errors.c
d6a6885c562c9944f4109d1db610c88c  ./usr/test/nameserver/test_nameserver_wrapper_errors.h
6faf50e64543eb99023ae87659405866  ./usr/test/test_message_benchmark.c
a78e59a86a563bbce35d5f4f310f0bc0  ./usr/test/test_message_benchmark.h
66e5b0427882e79c2d53d56325048470  ./usr/test/test_runner.c
6d81e23aee4ea81d63606fd28bf9759a  ./usr/test/test_runner.h
e22f4b33d4b2703e4fb73230ed9d8195  ./usr/test/test_undefined_handler.c
8bbd9e31e2e699efd055a1c4cbf6baa5  ./usr/test/test_undefined_handler.h
af9cdf9014bec08c6df74cead3ca105d  ./usr/track/README.txt
7707b57cb3a6c3703c0e3da3acbef763  ./usr/track/parse_track.py
1e06c8505279ed9f4d10136465c81618  ./usr/track/parts_tracka
a408ef1736b356024bf0dc2bc05d98b0  ./usr/track/parts_trackb
d38663c8897d9b202ff02e75b085197c  ./usr/track/track_data.c
8039d21694d3741e459dca0fb9b7eaa0  ./usr/track/track_data.h
d35c52f00718ac0aec535896687e2813  ./usr/track/track_node.h
50e0b1150b39a2425cb2180c33e0e57f  ./usr/track/tracka
dab2764f1d7f07aa454bad2ec01158b8  ./usr/track/trackb
6cae62c97dc3e1b1de204974a295e55a  ./usr/user_data_abort.h
a7668b81a997cb9d602415320f5d33f3  ./versatilepb.ld
3f052cb7f6b25c5e80dca05f284701d3  ./versatilepb_e2e.ld
```