Documention for the StrombolOS™ Real-Time Operating System

Authors:
Jacob Parker
Kyle Spaans

Contents

-3 Administrivia

A copy of our code can be found in `/u/j3parker/cs452/kernel/k3/`. The `doc/` directory includes our documentation, library etc. folders (our only library is the given bwio library.) The following is an MD5 sum listing for our files at the time of submission. Only sums for files *.h, *.s, *.c files.

```
db44d6b35a8af768aab33e6e521108a7  ./bwio/bwio.c
d32dda3f6cd59b210c03d1ed8332c581  ./include/bwio.h
a36a5cd3f6b82a3acb0c0155e1e0df6d  ./include/debug.h
1282c25ddcb52b0d29a5dbda8bac0de0  ./include/ts7200.h
782bb4b61ff4572a6306d3630e375b03  ./kernel/kernel.c
60e2d0e5c497824f2da2ba022dda3750  ./kernel/switch.h
1df8741f0738055aacee55341567c5b2  ./kernel/syscalls/create.c
4c445abfc7bbc947523149aa4ff8db69  ./kernel/syscalls/exit.c
88de93ad80f82c0c19068db925aa9a7b  ./kernel/syscalls/ksyscall.h
1fc352c47457bcc3483baf673453ca47  ./kernel/syscalls/myparenttid.c
7aea0293ec3376f3b609444439e0ffd8  ./kernel/syscalls/mytid.c
02a046f81c0b15aae6672817ba9c8fd7  ./kernel/syscalls/pass.c
b685aa17a204c0345d538968ef106cf2  ./kernel/syscalls/receive.c
5c08c82b4d0a9b6423c63ffcab50b7ed  ./kernel/syscalls/reply.c
5ee91374be2004fb41046d7eaa9b6605  ./kernel/syscalls/send.c
b8418a9a114d7785e9e01e94f8a8ad38  ./kernel/syscalls/awaitevent.c
68b6656bcc4edd225acd2abcdca7f24a  ./kernel/tasks.c
cf322c58d2b0ffca7960adb1731fc034  ./kernel/tasks.h
a7af4f1ca5ce0125d86ffaab2a0960b5  ./kernel/boot.c
b3601b4e827fe73ea426af25d82e0f1d  ./kernel/boot.h
63a9c591765b838b80d156ba4d343ea9  ./kernel/switch.s
3262fc7b205c0e480d153eaf844fbfb9  ./servers/names.c
543b0ddeef7d66f66b7126efa9b58ea0  ./servers/rand.c
53eb72a04263b25510535132b3453e35  ./servers/rps_server.c
072d5490d6b56708b35b21d338a2a301  ./servers/servers.h
598f8587f424025409a3b38b148a62ec  ./servers/clock.c
9daae8dbd35f90b91ba178bd8f4c55c5  ./servers/notifier_clock.c
bf165d3c3f5a63a7678f0f2e993d7dae  ./test/iotest.c
68b329da9893e34099c7d8ad5cb9c940  ./test/iotest.h
24f7ded4f1893ec68eaf5f81a66f3f03  ./user/lib.c
09ed0453640775e833aac93d47f63c9e  ./user/lib.h
bb26439fa9e96b4ad416cca1ffbee9d8  ./user/rps_client.c
394743b7c269f3b4eabc822e875fb126  ./user/user.c
d7f6b7b2bb5a4b987338d00760c7cd6a  ./user/user.h
32f09b7215980fca2c21d19757ee60b8  ./user/usyscall.h
3e319c32cd896bb5a3c2c9ee82bbb459  ./user/usyscall.s
8cffb55e1add4a3e42efee500d1b216c  ./user/wrappers.c
0b234aa42c2633669bc903a8385bedb2  ./user/clock_client.c
0a4ecd61c17f8fab2701df0fc602f52b  ./user/clock_client.h
2ec8ab7307e18e117f38649d0b503500  /u/cs452/tftpboot/ARM/StrombolOS/k3
```

-2 Sample Output

```
Please select an option (1:rps, 2:srr_tests, 3:clock): 3
[ :) ] .5.5.5.Child 5 of parent 4 created, asking for info...
[ :) ] --- Parent 4 created child 5 with priority 1
[ :) ] --- Parent 4 created child 6 with priority 2
[ :) ] .6.6.6.Child 6 of parent 4 created, asking for info...
[ :) ] --- Parent 4 created child 7 with priority 3
[ :) ] --- Parent 4 created child 8 with priority 4
[ :) ] --- Parent received from child 5 retval 0
[ :) ] .5.5.5.Child 5 delay for 10 ticks 20 times
[ :) ] TID 5 loop 0 of 20
[ :) ] --- Parent received from child 6 retval 0
```

```
[ :) ] .6.6.6.Child 6 delay for 23 ticks 9 times
[ :) ] TID 6 loop 0 of 9
[ :) ] .7.7.7.Child 7 of parent 4 created, asking for info...
[ :) ] --- Parent received from child 7 retval 0
[ :) ] .7.7.7.Child 7 delay for 33 ticks 6 times
[ :) ] TID 7 loop 0 of 6
[ :) ] .8.8.8.Child 8 of parent 4 created, asking for info...
[ :) ] --- Parent received from child 8 retval 0
[ :) ] --- Parent 4 exiting!
[ :) ] .8.8.8.Child 8 delay for 71 ticks 3 times
[ :) ] TID 8 loop 0 of 3
[ :) ] TID 5 loop 1 of 20
[ :) ] TID 5 loop 2 of 20
[ :) ] TID 6 loop 1 of 9
[ :) ] TID 5 loop 3 of 20
[ :) ] TID 7 loop 1 of 6
[ :) ] TID 5 loop 4 of 20
[ :) ] TID 6 loop 2 of 9
[ :) ] TID 5 loop 5 of 20
[ :) ] TID 5 loop 6 of 20
[ :) ] TID 7 loop 2 of 6
[ :) ] TID 6 loop 3 of 9
[ :) ] TID 8 loop 1 of 3
[ :) ] TID 5 loop 7 of 20
[ :) ] TID 5 loop 8 of 20
[ :) ] TID 6 loop 4 of 9
[ :) ] TID 5 loop 9 of 20
[ :) ] TID 7 loop 3 of 6
[ :) ] TID 5 loop 10 of 20
[ :) ] TID 6 loop 5 of 9
[ :) ] TID 5 loop 11 of 20
[ :) ] TID 5 loop 12 of 20
[ :) ] TID 7 loop 4 of 6
[ :) ] TID 5 loop 13 of 20
[ :) ] TID 6 loop 6 of 9
[ :) ] TID 8 loop 2 of 3
[ :) ] TID 5 loop 14 of 20
[ :) ] TID 5 loop 15 of 20
[ :) ] TID 6 loop 7 of 9
[ :) ] TID 7 loop 5 of 6
[ :) ] TID 5 loop 16 of 20
[ :) ] TID 5 loop 17 of 20
[ :) ] TID 6 loop 8 of 9
[ :) ] TID 5 loop 18 of 20
[ :) ] ================= 7 Exiting =================
[ :) ] TID 5 loop 19 of 20
[ :) ] ================= 6 Exiting =================
[ :) ] ================= 8 Exiting =================
[ :) ] ================= 5 Exiting =================
```

The task with TID 5 is set at a higher priority than the first user task, so it will run first whenever possible. TID 6 is the same priority as the first user task, and 7 and 8 are both lower. The rest of the output is randomness from the timer and simple alternation between each delay cycle. TID 8 finishes first, but is at too low of a priority to exit before TID 7 or TID 6 can. 7 is naturally first because it has fewer cycles than 6. This leaves 5, with the most highest number of calls to delay (even though it has a median number of cycles to complete: 20×5) and thus the largest overhead in spite of its high priority to finish last.

The output of the binary you will run is slighly different. There is a taskbar with the tick count, a welcome screen and the output of bootstrap and the first user task. You will need to press a key to start the clock user tasks that provide the above output. The kernel will stay in a while loop at the end. This is because we have disabled the command prompt for this assignment. The command prompt can exit to redboot or reboot (via turning on the watchdog timer.)

-1 README

This is from the README file in our project's `docs/` directory. This entire document is also available as `DOCS/overview.pdf`.

```
To launch the executable from RedBoot, type "l ARM/StrombolOS/k3; g" and
it will load and run.

It is required that the ts7200 be reset before running this kernel due to
pending interrupts from other kernels. (This is easy to fix and we will do
so in future assignments but it is still best to reset to get a garunteed
state for all the hardware registers.)
```

0 Kernel Description

0.1 AwaitEvent

For `AwaitEvent()`, we chose for interrupts to be on when the notifier is called and for only one task to be able to wait on any single event. Therefore the event queue can be represented by a simple array of pointers to TD structs. When a task is waiting on the event a pointer to its TD struct will be in the queue. The pointer is removed when the task is woken up. For the delay queues in `Delay()`, two seperate mechanisms are needed. First, to support $O(n)$ tasks being delayed at once, `MAXTASKS` linked list nodes are allocated statically. These are then setup as a ring buffer of "dynamically allocatable" nodes. As a task gets delayed, a list node is allocated and insertion-sorted into the list of delayed tasks. Sadly this takes $O(n)$ time. The upside is that the list is sorted such that the tasks that need to be woken up first are at the front of the list. This means that when the clock ticks, `Delay` can compare the current time and the wakeup time of the first node in the delay list and pop it off accordingly, possibly popping multiple delayers. The downside to this allocation scheme is that if there is a single long-delaying task, and many shorter delaying tasks, we may wrap around the ring buffer of list nodes and reusing the long sleeping tasks's list node. Fortunately, with `MAXTASKS` set to 50, it is very unlikely that the first three children will be able to wrap around and reuse the last child's node – the child who sleeps for 71 ticks.

We intend to test other data structures for the next assignment. The amount of tasks we were testing for this assignment were not enough to make any of our data structures consume anything more than a negligible amount of time.

0.2 IRQ Handling

The context switcher was rewritten from scratch for this assignment. activate starts a user task and through careful use of the `movs` and `LDM/STM` version 3 (load/store from/to user mode registers from supervisor mode) instructions we change to user mode in an atomic fashion. This is to prevent an edge case where an interrupt occurs before all of the user state is restored. We `movs pc,lr` on line x. At line $x + 1$ we have our IRQ handler. This carefully changes to supervisor mode (making sure to transport `SPSR_irq` and `lr_irq`.) A magic byte is stored in a magic location. Now the code "falls through" to the regular system call handler. We process things as if it was a regular system call (this is where our code goes on such a system call.) This code passes the value stored at the magic location to the kernel and sets it to 0. The kernel can tell that it is processing an interrupt by whether the magic byte is 0 or not.

The context switch is now under 20 instructions of code for a regular system call, so efficiency has been greatly improved.