CS 452 Kernel 2

Brian Forbes & Daniel McIntosh
20617899 & 20632796

June 5, 2018

1 Building, Running, and Usage

The repository for the code can be found at `https://git.uwaterloo.ca/baforbes/cs452-kernel/tree/k2`. To build the executable, run the following command from the root directory:

```
make
```

To build this document, run the following command from the root directory:

```
make docs
```

The executable will be built to `<repo_root>/bin/kernel.elf`

Once run, the program will initialize the kernel, and then create the first user task. This task will then create 3 other tasks: the Rock Paper Scissors (RPS) server, and 2 RPS clients. The clients will play 10 RPS games against each other, awaiting user input when they recieve replies from the RPS server so the game results are displayed.

2 Program Description

Note that each referenced file corresponds to a header file of the same name in the `include` folder (for example, `src/kernel.c` to `include/kernel.h`)

2.1 Context Switch

To facilitate interrupts, the context switch was updated to now push all user registers onto the stack (except PC and SP, which are saved separately). As a part of this, syscall arguments are now saved on the user stack as well (they are pushed as registers r0-r3, with the fifth argument pushed onto the stack before the syscall. Previously, these were being saved into the task struct.

A second kernel entry point has been added to `asm/activate.s` for interrupt requests: `IRQ_ENTRY_POINT`. It saves the value 1 in the IRQ Stack Pointer. Later

in the context switch, this value is checked. If the syscall is an IRQ, the IRQ LR is saved before returning to the kernel. If it is not an IRQ, the SWI Lr is saved instead, as previously.

2.2 IRQ Handling

2.3 Clock Server

The clock server is implemented in `src/clock.c`

It stores the current time (in 10ms ticks), and a minheap of the TIDs of tasks that have called delay. After initializing this data, it spawns a Notifier process. The notifier awaits the clock interrupt event, and then sends a notification to the clock server.

Once the clock server has finished both of these initialization steps, it begins an infinite loop of receiving, and then handling messages. There are four different message types:

NOTIFIER Upon reception, the clock server replies to the notifier task, and adds a tick to the current time. It then notifies all tasks which have delay deadlines which end before the current time, by replying to them.

TIME The clock server replies to the requesting task with the current time.

DELAY The clock server adds the task and the deadline (calculated as current time + delay time) to the minheap of waiting tasks.

DELAYUNTIL The clock server adds the task and the deadline (provided by the requester) to the minheap of waiting tasks.

2.4 Idle Time

2.5 Changes from Kernel 2

The only major change to the Kernel 2 architecture was the updates to `asm/activate.s`, as described in the Context Switch section.

2.6 New Data Structures

A minheap was used in the clock server to effeciently store the waiting tasks. It was implemented on top of a constant-size array. It has three operations: `add`, `remove_min`, `peek_min`.

add The element is added to the end of the array, and then bubbled upwards. This is logarithmic in the size of the array, which is a constant - so this takes constant time.

peek_min The element at the 0^{th} index is returned. This takes constant time.

remove_min The element at the 0^{th} index is swapped with the last element, and then removed. The new 0^{th} element is bubbled down. The removed element is then returned. Similarly to `add`, this is logatirhmic in the size of the array, which is still constant.

We chose a minheap for this purpose because it was more efficient compared to a sorted linked list, and more simple to implement without dynamic memory allocation. We felt as though implementing a pool of linked list nodes was more complex and error-prone than our minheap implementaion.

3 Explanation of Program Output

In the top-left corner, the idle time is displayed. The rest of the program outputs lines of the form:
`$(TID): $(DELAY). $(I)/$(N)`
Where `$(I)` is the current number of delays this task has completed, and `$(N)` is the total number of delays for this task.

4 Known Limitations

At the moment, the program cannot be run twice in a row without resetting the ARM box.

5 File Hashes and Commit hash

Output of `sha1sum src/* include/* asm/*`

Hash	File