Kernel 2

CS452 - Spring 2014

Real-Time Programming

Team
Max Chen - mqchen
mqchen@uwaterloo.ca

Ford Peprah - hkpeprah
ford.peprah@uwaterloo.ca

Bill Cowan
University of Waterloo

Due Date: Friday, 30^{th}, May, 2014

Table of Contents

1 Program Description

1.1 Getting the Program

To run the program, one must have read/write access to the source code, as well as the ability to make and run the program. Before attempting to run the pogram ensure that the following three conditions are met:

- You are currently logged in as one of `cs452`, `mqchen`, or `hkpeprah`.
- You have a directory in which to store the source code, e.g. `~/cs452_microkern_mqchen_hkpeprah`.
- You have a folder on the FTP srever with your username, e.g. `/u/cs452/tftp/ARM/cs452`.

First, you must get a copy of the code. To to this, log into one of the aforementioned accounts and change directories to the directory you created above (using `cd`), then run one of

```
git clone file:////u8/hkpeprah/cs452-microkern -b kernel2 .
                           or
git clone file:////u7/mqchen/cs452/cs452-microkern -b kernel2 .
```

You will now have a working instance of our kernel2 source code in your current directory. To make the application and upload it to the FTP server at the location listed above (`/u/cs452/tftp/ARM/YOUR_USERNAME`), run `make upload`.

1.2 Running the Program

To run the application, you need to load it into the RedBoot terminal. Ensure you've followed the steps listed above in the "Getting the Program" settings to ensure you have the correct directories and account set up. Navigate to the directory in which you cloned the source code and run `make upload`. The uploaded code should now be located at

```
/u/cs452/tftp/ARM/YOUR_USERNAME/assn2.elf
```

To run the application, go to the RedBoot terminal and run the command

```
load -b 0x00218000 -h 10.15.167.4 ''ARM/YOUR_USERNAME/assn2.elf''; go
```

The application should now begin by running through the game tasks before reaching a prompt. The generated files will be located in `DIR/build` where `DIR` is the directory you created in the earlier steps. To access and download an existing version of the code, those can be found at `/u/cs452/tftp/ARM/mqchen/assn2.elf` and `/u/cs452/tftp/ARM/hkpeprah/assn2.elf`.

2 Kernel Structure

2.1 System Calls

The following system calls have been implemented in this assignment:

System Call	Prototype	Description
RegisterAs	`RegisterAs(char *name)`	Register the current task with the name `name` in the NameServer
WhoIs	`WhoIs(char *name)`	Lookup the task with the given name in the NameServer and returns its tid if found
UnRegister	`UnRegister(char *name)`	Unregister the current task iff its name and tid match what is in the NameServer

3 Game Tasks

3.1 Priorities

Task Name	Task ID	Priority
FirstTask	0	15
NameServer	1	15
Server	2	11
Client (NXCLZ)	3	7
Client (HIIJS)	4	8
Client (JUWKR)	5	5
Client (YEOYF)	6	1
Client (NLZEM)	7	7
Client (GXKFQ)	8	3
Client (LPEKV)	9	1
Client (CXWTY)	10	6
Client (ABJFT)	11	8
Client (TXRHX)	12	6

The priority for the FirstTask (the task that bootstraps the NameServer, Server, and Clients/-Players) was chosen to be 15 because 15 is the highest possible priority supported by our kernel, thus allowing the task to proceed without yielding to another task, self the Name-Server which blocks on `Receive`, to allow it to create all the tasks as soon as possible before exiting. The priority of the NameServer was chosen to be 15 to ensure that the NameServer was created and did any of the necessary setup work before any other task that would need it was to be run. This ensure that any task calling `RegisterAs` or `WhoIs` would succeed, as the NameServer `RCV_BLK`s (Receive Blocks) as the last step in its setup. So before any task needs the NameServer, it is already setup and waiting to be sent messages. The priorities of the Server and the Clients are unimportant with respect to one another, as any client with a higher priority than the server would just block waiting for the server to respond to it; the priorities were essentially random to allow for diversity in the result. The only important factor was to ensure that the priorities were less than the NameServer to ensure that the NameServer was ready before they would need it. There were two different approaches to do this; have the first task send a message to the nameserver, blocking and allowing the nameserver to setup and respond before creating the remaining tasks, or have the other tasks lower priority so that the NameServer was the next task scheduled after the first task exited; we chose the latter approach. Since the tasks `SEND_BLK` when they message the server, the server's priority does not matter, as they will be put on its queue when its execute to avoid a task with higher priority looping until it is given a response.

3.2 Game Task Output

The output from the GameTask is as follows:

```
Player HIIJS(Task 4) throwing PAPER
Player ABJFT(Task 11) throwing ROCK
HIIJS won the round
Press any key to continue:

Player NXCLZ(Task 3) throwing PAPER
Player NLZEM(Task 7) throwing SCISSORS
NLZEM won the round
Press any key to continue:

Player CXWTY(Task 10) throwing ROCK
Player TXRHX(Task 12) throwing SCISSORS
CXWTY won the round
Press any key to continue:

Player JUWKR(Task 5) throwing SCISSORS
Player GXKFQ(Task 8) throwing SCISSORS
Round was a TIE
Press any key to continue:

Player JUWKR(Task 5) throwing ROCK
Player GXKFQ(Task 8) throwing PAPER
GXKFQ won the round
Press any key to continue:
```

```
Player YEOYF(Task 6) throwing SCISSORS
Player LPEKV(Task 9) throwing ROCK
LPEKV won the round
Press any key to continue:
```

The implementation of `random` using a set seed, so the results from the game are deterministic, which allows us to argue that the results will always be the same as above. First, the explanation of how Rock-Papers-Scissors works. To begin a game of Rock-Paper-Scissors, two parties must agree to play, at which point, each party throws one of {Rock, Paper, Scissors} simulataenously. Rock beats Scissors, Scissors beats Paper, and for some god awful reason, Paper beats Rock. If both parties throw the same hand, the round ends in a tie, and neither party is victorious. Task 4 and 11 have the highest priority of the client/player tasks, so they are the two first to run. They both signup with the server by sending a SIGNUP request, and give their hands by sending a PLAY request. Since task 4 threw PAPER and task 11 threw ROCK, task 4 wins the round, and they both send a QUIT request if the result is not a tie. The result of the game is sent back as a reply to the two players indicating if they won, if they lost, or if it was a tie. Any tasks that make a request to PLAY while a game is in session, is queued by the server and replied to when there is a free slot for that task to join. The next two tasks then run, and so on, in order of decreasing priority. On a tie, such as in the first case of task 5 and task 8, they play again by sending another PLAY request. So, the order of the tasks by priority is {4, 11, 3, 7, 10, 12, 5, 8, 6, 9}, which takes into account the order the tasks were created. So 4 plays against 11, 3 against 7, 10 against 12, 5 against 8, and 6 against 9 as this is the order in which the tasks run and signup, and the first two tasks to signup are the first two to play; the server sends a reply indicating they can now play and to send it their hands. After each round, the server prints out the tasks in order of when they signed up, what they threw, and the result of the result, then prompts the actual user for input to continue execution.

4 Performance Measurements

4.1 Results

Message Length	Caches	Send Before Receive*	Optimization	Microseconds
4 bytes	off	yes	off	343.8453713
64 bytes	off	yes	off	462.8687691
4 bytes	on	yes	off	24.41505595
64 bytes	on	yes	off	31.53611394
4 bytes	off	no	off	378.4333672
64 bytes	off	no	off	496.439471
4 bytes	on	no	off	27.46693795
64 bytes	on	no	off	35.60528993
4 bytes	off	yes	on	192.2685656
64 bytes	off	yes	on	231.9430315
4 bytes	on	yes	on	12.20752798
64 bytes	on	yes	on	15.25940997
4 bytes	off	no	on	215.6663276
64 bytes	off	no	on	255.3407935
4 bytes	on	no	on	14.24211597
64 bytes	on	no	on	16.27670397

***** - Assignment says “Send Before Reply”, however, replies are non-blocking and don’t depend on a send to occur.

4.2 Explanation

Something something...

5 MD5 Hashes

Source files can be accessed at either `/u7/mqchen/cs452/cs452-microkern` or `/u8/hkpeprah/cs452-microkern`. The MD5 hashes of the source files are as follows: