CS452 – SPRING 2010
KERNEL 2

BILL COWAN
UNIVERSITY OF WATERLOO

A. INTRODUCTION

In the second part of your development of the kernel, you make it possible for tasks to communicate by message passing. You use this capability to create a name server. To do so you need to have working

1. implementations of Send, Receive and Reply,
2. a running name server, created by the first user task, and
3. implementations of WhoIs and RegisterAs as wrappers for Send to the name server.

As discussed in class you must choose a method by which every task knows the task id of the name server, which is its bootstrap into the world of services provided by servers closely associated with the kernel.

In addition to the kernel primitives you must program a server and some clients. The server is the custodian of a Rock/Paper/Scissors game; the clients play the game against one another by interacting with the kernel. Finally, we would like you to measure the time (in clock cycles or in seconds) for Send/Receive/Reply transaction for two sizes of message: four (4) bytes and sixty-four (64) bytes.

Comment. It is not uncommon to discover weaknesses in the first part of the kernel as you work on the later parts. So it is good strategy to start early.

B. DESCRIPTION

B.1. KERNEL

To accomplish this part of the kernel you must have the following kernel primitives operating:

- `int Send( int tid, char *msg, int msglen, char *reply, int replylen )`,
- `int Receive( int *tid, char *msg, int msglen )`, and
- `int Reply( int tid, char *reply, int replylen )`.

See the kernel description and the lecture notes for the details of how these primitives should operate.

In addition you must program a first user task, which

- creates the name server,
- creates the Rock/Paper/Scissors server, and
- creates the Rock/Paper/Scissors clients.

B.2. User tasks.

The following user tasks test your kernel and name server. They should be created by the first user task after it has initialized the system.

B.2.I. *Rock/Paper/Scissors (RPS) Server*

The RPS server should accept and service the following three types of request.

- Sign up. Sign up requests are sent by clients that wish to play. They are queued when received, and when two are on the queue the server replies to each, asking for the first choice.
- Play. Play requests tell the server which of Rock, Paper or Scissors the choose on this round. When play requests have been received from a pair of clients, the server replies giving the result.
- Quit. Quit tells the server that a client no longer wishes to play. The server replies to let the client go, and responds to next play request from the other client by replying that the other player quit.

B.2.II. *Rock/Paper/Scissors Clients*

Clients that play the game should

- find the RPS server by querying the name server,
- perform a set of requests that adequately tests the RPS server,
- send a quit request when they have finished playing, and
- exit gracefully.

The game should pause at the end of every round of the game so that the TA can see what happened. `bwgetc` is very handy for this.

Unless the opposing player is very stupid, a client cannot do better than playing randomly.

B.3. Performance Measurement

Create two tasks that exchange messages using Send/Receive/Reply. Measure the time from the first instruction of the Send wrapper code to the last instruction before the wrapper code returns for 64 bytes messages and 4 byte messages. (By x byte message I mean that there are x bytes sent and x bytes replied.)

There should be no code between Receive and Reply in the receiving task.

C. Hand in

Hand in the following, nicely formatted and printed.

1. A description of how to operate your program, including the full pathname of your executable file which we will download for testing.

2. A description of the structure of your kernel so far.* We will judge your kernel primarily on the basis of this description. Describe which algorithms and data structures you used and why you chose them.
3. The location of all source code you created for the assignment and a set of MD5 hashes of each file. The code must remain unmodified after submission until the assignments are returned.
4. A listing of all files submitted.
5. A short description of what priorities you chose for the game tasks, and why you chose them.
6. The measurements you made, and a brief explanation of where in your code you think the time is being spent.
7. Output produced by your game task and an explanation of why it occurs in the order it does.

---

* You should be accumulating your hand in documents as you go along because we expect a description of your complete kernel when its development is complete. But now hand in only a description of what is new or changed.