CS 452 Kernel 2

Benjamin Zhao, Kyle Verhoog

Jan 29, 2018

Contents

Operation

ELF location

The ELF file is located at `/u/cs452/tftp/ARM/yl4zhao/kernel2.elf` and can be initiated using the typical `load` command.

After loading the kernel should execute the sample test required for the assignment.

Implementation

Messaging

Improvements on the implementation of the kernel affects the `handle` method.

```
void handle(TaskDescriptor *td, TaskRequest req) {
  switch (req) {
  ...
  case SEND:
    send_handler(td);
    ...
  case RECEIVE:
    receive_handler(td);
    ...
  case REPLY:
    reply_handler(td);
    ...
  ...
  }
}

void send_handler(TaskDescriptor *std);
void receive_handler(TaskDescriptor *rtd);
void reply_handler(TaskDescriptor *rtd);
```

Data Structures and Algorithms

The implementation follows similar concepts described in class. Note the Syscall registers r0 to r3 are saved to the user stack on context switch. The updated Task Descriptor now has a queue holding blocked sender tids.

`send_handler` retrieves the r0-r3 arguments from the sender's stack and copies the msg to the receiver.

`receive_handler` retrieves r0-r2 arguments from receiver's stack, copies if sender is available, otherwise queue's sender.

`reply_handler` retrieves r0-r2 arguments from receiver's stack, and copies reply msg to the sender.

Syscalls

New SWI calls have been added for user tasks to make communication requests to other tasks.

```
int Send(int tid, void *msg, int msg_len, void *reply, int reply_len);
int Receive(int *tid, void *msg, int msg_len);
int Reply(int tid, void *reply, int reply_len);
```

NameServer

A special task which serves as a nameserver for all tasks to set and get information about others.

```
int RegisterAs(int n);
int WhoIs(int n);
void NameServerTask();
```

Data Structures and Algorithms

The Nameserver uses a basic `<int, int>` mapping from an integer name to a tid with a fixed size array, O(1) time. Tasks can register their own tid to a static integer name using `RegisterAs`. Tasks can also query the tid of a static name from the nameserver using `WhoIs`. The NameServer service can be stopped by sending the Stop command. Note: Only the curator of the NameServer has the ability to stop it (as it is the only one who directly has access to the NameServer's Tid). Note, when the NameServer initializes, it calls `RegisterNS` which registers its Tid to the kernel (for flexibility on NameServer defines).

```
typedef enum NSservice {
    WhoIs_t = 0,
    RegisterAs_t = 1,
    Stop_t = 2
} NSservice;
```

`RegisterAs` is a special Syscall function which queries the tid of the registered NameServer, then calling the `Send` syscall with pre-filled parameters and expects the NameServer to return a success or failure on registering the user task into the nameserver.

`WhoIs` is a special Syscall which queries the tid of the registed NameServer, and calling the `Send` syscall with pre-filled paramets and expects the NameServer to return the associated Tid to the name.

RPS Game

A Rock-Paper-Scissor Client/Server Test on the communication with the kernel. (Note: We named acronym RPS as RPC (Rock-Paper-sCissor) by accident.)

```
void RPCClient();  \\Plays a move based on Tid
void RPCClient2(); \\Always quits
void RPCServer();  \\Server handles ONLY ONE GAME AT A TIME (All other players are queued)
```

Data Structures and Algorithms

The RPS Client/Server tasks communicate with each other to play the game. Clients signup to the Server, which are then queued and paired if two or more clients have signed up. Note that only one game can happen at a time, so all subsequent players who signed up are queued until the first pair has finished playing. One in every 3 players spawned is toxic and will quit the game when paired.i Every other player will make a move based on their Tid mod 3 (Rock - 0, Paper - 1, Scissor - 2). A total of 50 players are created, (starting from tid 3), thus 25 games are played.

```
typedef enum RPCservice{
    S_Signup = 0,
    S_Play = 1,
    S_Quit = 2,
    S_Close = 3
}RPCservice;
```

`Signup` is a service call to the server to queue the client up for play. Once matched, the server will reply to both players.

`Play` is a service call to the server to play the move of their choice. Once both players have made their move (or quit), the server will reply to both players with the outcome.

`Quit` (same as play).

`Close` is a service call to shutdown the server.

Msg Length	Caches	Send before Reply	-O2	Time (μs)
4	off	yes	off	354.00391
64	off	yes	off	846.35417
4	on	yes	off	25.43132
64	on	yes	off	59.00065
4	off	no	off	325.52083
64	off	no	off	817.87109
4	on	no	off	23.396810
64	on	no	off	56.966146
4	off	yes	on	188.19173

Msg Length	Caches	Send before Reply	-O2	Time (μs)
64	off	yes	on	329.58984
4	on	yes	on	12.20703
64	on	yes	on	22.37956
4	off	no	on	224.81283
64	off	no	on	316.36556
4	on	no	on	11.18978
64	on	no	on	21.36230