ECE 454 Assignment 1 Test Tool Manual

June 20, 2014

Setup

1. unzip the content of tetstool.zip into your workspace where your **makefile** resides
2. make sure that all the required files mentioned below are present
3. simply run the command `sh makeAll.sh`
4. check to see if **server.out** and **client.out** are there

Required Files:

1. makeAll.sh
2. ece454.mk
3. ece454_client_app
4. ece454_server_app
5. ece454_test_functions

Usage

Running the Server

Run the following command

```
$./server.out [procname]*
```

Where:

```
procname=[proc1 | proc2 | proc3 | proc4 | proc5]
```

`proc1`, ..., `proc5` are simply aliases for various test function. This command registers those functions under the names `proc1`, ..., `proc5` accordingly. Here is a very brief description of these procedures:

- `proc1` is a procedure with 1 fixed size parameter and a fixed size return value
- `proc2` is a procedure with 2 fixed size parameters and a fixed size return value
- `proc3` is a procedure with 2 variable size parameters and a fixed size return value
- `proc4` is a procedure with 6 variable size parameters and a variable size return value
- `proc5` is a procedure with 0 parameters and a fixed size return value

Note that you can register a selection of these procedures in on run. For example:

```
$./server.out proc1 proc2 proc3 proc4 proc5
```

or

```
$./server.out proc4 proc2
```

However, do not use the same proc in the same run unless you code resolves procedure name conflicts.

Running the Client

Run the following command

```
$./client.out <server_address> <server_port> [procname]*
```

Where:

`server_address` = IP address printed by `./server.out` command

`server_port` = port printed by `./server.out` command

`procname=[proc1 | proc2 | proc3 | proc4 | proc5| noproc]`

The client application simply queries each one of the procs (provided on the command line) from the server and validates their return values. `proc1, ... , proc5` are the same procs described in the previous section. `noproc` instructs the client to query for an unregistered procedure and check whether the server responses appropriately.

As an example:

```
$./client.out someip someport proc1
```

will call `make_remote_call("someip", someport, "proc1", 1, ...)`, and checks the validity of servers response (given that `proc1` is registered on the server.) If the return value is correct, you should see a "proc1 successful" message on your terminal.

Note that like the server application, client application can query for a combination of procs, e.g:

```
$./server.out proc1 proc2 noproc proc5 proc2 proc3 ...
```

However, unlike the server , duplicate procs are acceptable.

Testing Suggestions and Scenarios

First and foremost, make sure that you instantiate your server and client on separate machines to make sure your communications are actually remote. Next make sure that `proc1 proc2 proc3 proc4` each pass individually since they are each a very basic scenario. Finally, you can try the following scenarios.

Scenarios:

1. scenario1:
   - **server side:** `$./server.out proc1`
   - **client side:** `$./client.out ip port proc1`
2. scenario2:
   - repeat scenario1 for `proc2 proc3 proc4 proc5`
3. scenario3:
   - **server side:** `$./server.out proc1 proc2 proc3 proc4 proc5`
   - **client side:** `$./client.out ip port proc1 proc2 proc3 proc4 proc5`
4. scenario4:
   - **server side:** `$./server.out`
   - **client side:** `$./client.out ip port noproc`
5. scenario5:
   - **server side:** `$./server.out proc1`
   - **client side:** `$./client.out ip port noproc`
6. scenario6:
   - **server side:** `$./server.out proc1`
   - **client side:** `$./client.out ip port proc1`
7. scenario7:
   - **server side:** `$./server.out proc1`
   - **client side:** connect to 2 "separate" machines (other than the servers machine) and run `$./client.out ip port proc1` on each