

# COMP 8005 Final Project Test Document

Christopher Eng

March 11, 2015

## port\_fwd

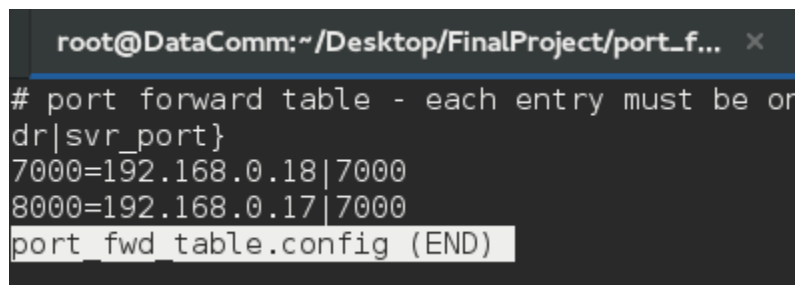
The following general program features will not be documented, as they were tested during development and are unrelated to the performance and core functionality of the program:

- Valid arguments
- Logging
- Configuration file
- Unit tests
- Epolling
- Socket networking (Connecting, sending, receiving)
- Thread parameters
- Cleanup: Closing FDs/freeing memory

Testing was done over a number of different machines over a LAN connection. It was made up of five different running programs:

- 1 machine hosted the port forwarder
- 2 machines each hosted an echo Epoll server
- 2 machines each hosted a TCP client

The following port forward table was used to connect each client to its own Epoll server.



```
root@DataComm:~/Desktop/FinalProject/port_f... x
# port forward table - each entry must be on
dr|svr_port}
7000=192.168.0.18|7000
8000=192.168.0.17|7000
port_fwd_table.config (END)
```

## Core Functionality

### Port Forward Table

While the configuration file format itself will not be tested, the receiving port must be read from the port forward table and have a destination IP address/port pair. From the port forward table, the ports 7000 and 8000 should receive messages from clients. The following screenshot is an excerpt using the “netstat -atn” command on the machine running the port forwarder. At this time, each TCP client was

transmitting requests to a different echo server. The local column shows at what ports the program is receiving on. Before the test was run, the tester made sure that no TCP connections existed.

root@DataComm:~/Desktop/FinalProject/port_f... x				root@DataComm:~/Desktop/Assignment2/tc		
tcp	0	0	192.168.0.21:8000	192.168.0.20:56360	ESTABLISHED	
tcp	0	0	192.168.0.21:44097	192.168.0.20:7000	ESTABLISHED	
tcp	0	0	192.168.0.21:7000	192.168.0.19:34304	ESTABLISHED	
tcp	0	0	192.168.0.21:42244	192.168.0.20:7000	ESTABLISHED	
tcp	0	0	192.168.0.21:46370	192.168.0.20:7000	ESTABLISHED	
tcp	0	0	192.168.0.21:39937	192.168.0.20:7000	ESTABLISHED	
tcp	255	0	192.168.0.21:8000	192.168.0.20:55842	ESTABLISHED	
tcp	0	0	192.168.0.21:42709	192.168.0.20:7000	ESTABLISHED	
tcp	0	0	192.168.0.21:38122	192.168.0.20:7000	ESTABLISHED	
tcp	0	0	192.168.0.21:7000	192.168.0.19:36049	ESTABLISHED	

Unit tests provided additional support that the port forward table configurations were properly being stored on the program and port-level file descriptors were added to the listening Epoll sets.

### Basic Port Forward Functionality

Basic port forward functionality is shown by creating a connection to the destination IP address/port on accepting a new client connection. In addition, the data received from the receiving port must be identical to that which is sent to the destination. The following diagram shows a screenshot of a locally run port forwarder and TCP client. In this case, we can see from debugging output statements that the data sent, "DATA", is the same from both the client and the port forwarder. The number of bytes sent (255) also matches.

```

ceng@pnutzubuntu: ~/Documents/SchoolProjects/8005/FinalProject/port_fwd
Sending: fd 39, request #1 - DATA
03/24/15 09:59:27:518 | 38 | 39 | 1 | 255
Sending: fd 41, request #1 - DATA
03/24/15 09:59:27:607 | 40 | 41 | 1 | 255
Sending: fd 43, request #1 - DATA
03/24/15 09:59:27:735 | 42 | 43 | 1 | 255
Sending: fd 45, request #1 - DATA
03/24/15 09:59:27:478 | 44 | 45 | 1 | 255
Sending: fd 47, request #1 - DATA
03/24/15 09:59:27:597 | 46 | 47 | 1 | 255
Sending: fd 49, request #1 - DATA
03/24/15 09:59:27:759 | 48 | 49 | 1 | 255
Sending: fd 51, request #1 - DATA
03/24/15 09:59:27: 1 | 50 | 51 | 1 | 255
Sending: fd 53, request #1 - DATA

ceng@pnutzubuntu: ~/Documents/SchoolProjects/8005/Assignment2/tcp_clnt
Transmit 0: DATA
Connected: Server Name: 127.0.0.1
Transmit 0: DATA
03/24/15 09:59:27:723 | 6 | 1 | 255 | 163498
03/24/15 09:59:27:885 | 7 | 1 | 255 | 164503
03/24/15 09:59:27:932 | 8 | 1 | 255 | 164652
03/24/15 09:59:27:974 | 9 | 1 | 255 | 164807
03/24/15 09:59:27:640 | 5 | 1 | 255 | 174162
03/24/15 09:59:27:814 | 4 | 1 | 255 | 174206
03/24/15 09:59:27:960 | 3 | 1 | 255 | 174199
03/24/15 09:59:27:104 | 2 | 1 | 255 | 174247
03/24/15 09:59:27:278 | 1 | 1 | 255 | 174257
03/24/15 09:59:27:423 | 0 | 1 | 255 | 174143
  
```

## Two-Way Communication

This function can be tested by proving that the server response is sent to the original client requester. Using an echo server to test makes this easy to show, since the server will respond with the same message and number of bytes. The following screenshot shows the output file from the port forwarder. As we can see, the initial client is being received on FD 38 and the port forwarder is relaying the data out FD 42. Several milliseconds later, we see the port forwarder has received a response from FD 42 and in turn, relays the message back to FD 38.

Time	ReceiveFD	SendFD	# Requests	Bytes Sent
03/23/15 22:46:38:231	38	42	1	255
03/23/15 22:46:38:572	41	44	1	255
03/23/15 22:46:38:948	46	47	1	255
03/23/15 22:46:38:986	44	41	1	255
03/23/15 22:46:38:996	57	60	1	255
03/23/15 22:46:38: 77	50	56	1	255
03/23/15 22:46:38:273	36	40	1	255
03/23/15 22:46:38:285	58	59	1	255
03/23/15 22:46:38:294	42	38	1	255
03/23/15 22:46:38:302	63	64	1	255

## Multiple Forwarded Ports

The program must be tested to prove that it operates properly when dealing with two different port forward connections at the same time. This will simulate the cases where there are more than two different port forward connections. As an extension to the previous screenshot, we now note the destination address to the different ports. In this single session, we see the program sends requests to port 8000 forward to 192.168.0.20 and requests to port 7000 are forwarded to 192.168.0.19. This was a different test from the port forward table configuration listed above, as you may notice the destination addresses are different.

root@DataComm:~/Desktop/FinalProject/port_f... x				root@DataComm:~/Desktop/Assignment2/tc	
tcp	0	0	192.168.0.21:8000	192.168.0.20:56360	ESTABLISHED
tcp	0	0	192.168.0.21:44097	192.168.0.20:7000	ESTABLISHED
tcp	0	0	192.168.0.21:7000	192.168.0.19:34304	ESTABLISHED
tcp	0	0	192.168.0.21:42244	192.168.0.20:7000	ESTABLISHED
tcp	0	0	192.168.0.21:46370	192.168.0.20:7000	ESTABLISHED
tcp	0	0	192.168.0.21:39937	192.168.0.20:7000	ESTABLISHED
tcp	255	0	192.168.0.21:8000	192.168.0.20:55842	ESTABLISHED
tcp	0	0	192.168.0.21:42709	192.168.0.20:7000	ESTABLISHED
tcp	0	0	192.168.0.21:38122	192.168.0.20:7000	ESTABLISHED
tcp	0	0	192.168.0.21:7000	192.168.0.19:36049	ESTABLISHED

## Closing Connection

When one end of the connection closes, the port forwarder must close the other side of the connection. This was tested by running two cases:

- 1) Closing an active connection from the client-side.

The following screenshot shows the port forwarder, TCP client, and Epoll server respectively. As we can see from the “^C” symbol at the TCP client, the connection was cut abruptly by the user. The connection from the port forwarder to the server was cut along with the client’s connection. The server continues to run.

```
03/24/15 10:55:38:522 |      41 |      40 |      3 |
03/24/15 10:55:38:665 |      45 |      44 |      3 |
03/24/15 10:55:38:796 |      43 |      42 |      3 |
03/24/15 10:55:38:925 |      39 |      38 |      3 |
03/24/15 10:55:38:353 |      37 |      36 |      3 |
03/24/15 10:55:38:398 |      49 |      48 |      3 |
03/24/15 10:55:38:435 |      51 |      50 |      3 |
03/24/15 10:55:38:505 |      53 |      52 |      3 |
03/24/15 10:55:38:542 |      47 |      46 |      3 |
Completed connection for server fd 34
Completed connection for client fd 35
Completed connection for server fd 36
Completed connection for client fd 37
Completed connection for server fd 38
Completed connection for client fd 39
03/24/15 10:55:38:750 |       4 |       3 |     765 |
03/24/15 10:55:38:879 |       7 |       3 |     765 |
03/24/15 10:55:38:315 |       5 |       3 |     765 |
03/24/15 10:55:38:383 |       2 |       3 |     765 |
03/24/15 10:55:38:421 |       1 |       3 |     765 |
03/24/15 10:55:38:491 |       0 |       3 |     765 |
03/24/15 10:55:38:529 |       3 |       3 |     765 |
^Cfeng@pnutzubuntu:~/Documents/SchoolProjects/8005/Assignm
03/24/15 10:55:38:657 |      40 |       3 |     765 |
03/24/15 10:55:38:735 |      41 |       3 |     765 |
03/24/15 10:55:38:796 |      38 |       3 |     765 |
Completed connection for fd 32
Completed connection for fd 33
Completed connection for fd 34
Completed connection for fd 35
Completed connection for fd 36
Completed connection for fd 37
Completed connection for fd 38
Completed connection for fd 39
Completed connection for fd 40
Completed connection for fd 41
```

## 2) Closing an active connection from the server-side.

The following screenshot is also in the order: port forwarder, TCP client, and Epoll server. This time, we see the server instance was closed during the port forward connection. The port forwarder closes both connections to the server and client.

```
Completed connection for server fd 48
Completed connection for client fd 51
Completed connection for server fd 50
Completed connection for client fd 53
Completed connection for server fd 52
[ ]

03/24/15 11:01:09:258 | 7 | 3 | 765
03/24/15 11:01:09:359 | 6 | 3 | 765
03/24/15 11:01:10:664 | 2 | 3 | 765
03/24/15 11:01:10:831 | 1 | 3 | 765
03/24/15 11:01:10:938 | 0 | 3 | 765
[ ]

03/24/15 11:01:09:604 | 38 | 3 | 765
03/24/15 11:01:09:666 | 33 | 3 | 765
03/24/15 11:01:09:728 | 32 | 3 | 765
03/24/15 11:01:09:637 | 39 | 3 | 765
03/24/15 11:01:09:638 | 40 | 3 | 765
03/24/15 11:01:09:639 | 41 | 3 | 765
^Ceng@pnutzubuntu:~/Documents/SchoolProjects/8005/Assign
```

## Varying Message Lengths

The Port Forwarder must be able to receive messages of any data length. The receive buffer was capped at 5000 bytes for the use case of the project, so this feature should show that the program can receive any amount under 5000 bytes.

For simplicity, the testing of this feature was split so that each run all clients sent messages of the same length. Messages of length 255 bytes and 4000 bytes were sent in two separate tests. The following screenshot shows a clip of the port forwarder output files under 4000 byte messaging and 255 byte messaging. The only change made during these tests was the buffer length argument for the TCP client. As we can see, the first sample has message lengths that are multiples of 4000 and the second sample has message lengths that are multiples of 255.

```
03/23/15 22:54:55:763 | 6402 | 6399 | 7 | 28000
03/23/15 22:54:55:176 | 160 | 159 | 8 | 32000
03/23/15 22:54:55:431 | 1368 | 1369 | 7 | 28000
03/23/15 22:54:55:473 | 7074 | 7075 | 6 | 24000
03/23/15 22:54:55:982 | 7075 | 7074 | 6 | 24000
03/23/15 22:54:55: 14 | 1344 | 1342 | 7 | 28000
03/23/15 22:54:55: 38 | 222 | 224 | 8 | 32000

03/23/15 22:46:45:859 | 15171 | 15167 | 4 | 1020
03/23/15 22:46:45:897 | 15360 | 15354 | 4 | 1020
03/23/15 22:46:45:936 | 2120 | 2118 | 8 | 2040
03/23/15 22:46:45:975 | 2184 | 2182 | 8 | 2040
03/23/15 22:46:45: 13 | 2159 | 2147 | 8 | 2040
03/23/15 22:46:45: 52 | 7956 | 7954 | 5 | 1275
```

## Pipe Communication

Thread communication is done through pipe connections. The two pipe instances are the output pipe, which send port forward information for file output, and the accept thread pipes, which transfer accepted connections from the accept thread to specific worker threads. This test is done while the program is not under heavy load, as we will see how the pipes work under heavy load in the performance section.

By showing that the pipe communication data is correct and that it is successfully read, the test for pipe communication is considered a success. The output pipe is easily shown, since the output thread is required to read from the pipe in order for any data to be written to the output file. The port forwarder displays a message when an accepted connection is written to a pipe, listing the new FD and which worker thread it was sent to. The worker thread also prints a message upon receiving these FDs. The order of the FDs down the pipes matter, since the clients then server FDs are sent and the worker thread expects the same order. Given that the accept thread is the only writer to the pipe and each pipe only has one reader thread, we can assume this order is thread-safe. The following screenshot shows the pipe messages upon the accept thread opening a new connection.

```
Remote Address: 127.0.0.1, 37
write to 0 pipe: 37
Remote Address: 127.0.0.1, 38
write to 0 pipe: 38
pipe 0 read new_fd 37
pipe 0 read new_fd 38
```

## Performance

In comparison to the Epoll server, the port forwarder must handle two concurrent connections for each client request, one connection to the client and another connection to the associated server. In the same way, the port forwarder must send and receive double the amount of data, having to relay each request. The Epoll server was able to handle 65000 concurrent connections sending payloads of up to 4000 bytes per request. However, with 60000 concurrent requests, the port forwarder struggled to keep up with payloads of 255 bytes.

By utilizing the TCP client's echo time (microseconds), we can compare the performance difference between the Epoll server and the port forwarder. Of course we expect the port forwarder to perform slower than the Epoll server, since the Epoll server is being used in the test and adds its own echo time to the count.

### 255 byte payload – 5 second interval at peak concurrency

Number of Concurrent Connections (Epoll Server)	Average Echo Time / Second (Epoll Server)	Number of Concurrent Connections (Port Forwarder)	Average Echo Time (Port Forwarder)
25000	69928.35	16000	6674.76

26000	59495.58	20000	9958.79
27000	49823.75	24000	39289.93
28000	61173.86	32000	30359.30
56000	55054.26	40000	48124.89
65525	97304.50	60000	106472.24

#### **4000 byte payload – 5 second interval at peak concurrency**

Number of Concurrent Connections (Epoll Server)	Average Echo Time / Second (Epoll Server)	Number of Concurrent Connections (Port Forwarder)	Average Echo Time (Port Forwarder)
20000	59494.68	11866	59512.99
65525	59511.57	14142	60034.45
		16042	75128.33

As we can see from the data samples, the port forwarder performed worse than expected. The echo time for clients was quite a bit worse than double the number of concurrent connections on the Epoll server. The number of concurrent connections was measured using the “netstat -atn | grep ESTABLISHED | wc -l” command, to obtain a count of ESTABLISHED connections. Under a heavier load, the port forwarder could not handle more than 16000 connections. The rate at which requests were being handled was much lower than during regular runtime.

A weakness to the pipe design was that under heavy payloads, the pipes tended to fill up before they could be read. This caused the pipe writers to wait for room on the pipe and heavily delay processing. This applied both to outputting data and adding new connections from the accept thread. Using 4000 byte payloads, the port forwarder was able to handle 16000 connections. However, when testing more concurrent connections, the amount of output data decreased significantly, as the threads completed the requests as fast as they could. The number of concurrent requests measured for these tests came out to be 11866, 14142, and 16042 requests.