COMP 8005 Final Project Design Document

Christopher Eng

March 23, 2015

The goal of the Network and Security Applications Development Final Project was to develop a simple “Port Forwarder”. The general concept of a port forward application is as middle-ware between a client and server. The client would connect to the port forwarder using a specific port. The port forwarder has a number of pre-set port configurations, each mapped to a destination server and port. The client’s connection to the port forwarder would in turn create a connection between the port forwarder and the destination server. All data sent from the client would be forwarded to the server. The port forwarder would also handle forwarding responses from the server back to the client.

For the purposes of this project, the program was restricted to only TCP connections. A simple way to break down the design of the project, as well as implementing an efficient way of handling many concurrent connections, was to look back at the multi-threaded epoll system call-based echo server designed in COMP 8005 Assignment 2. Refer to Figure 1 for a sequence diagram of the echo server.

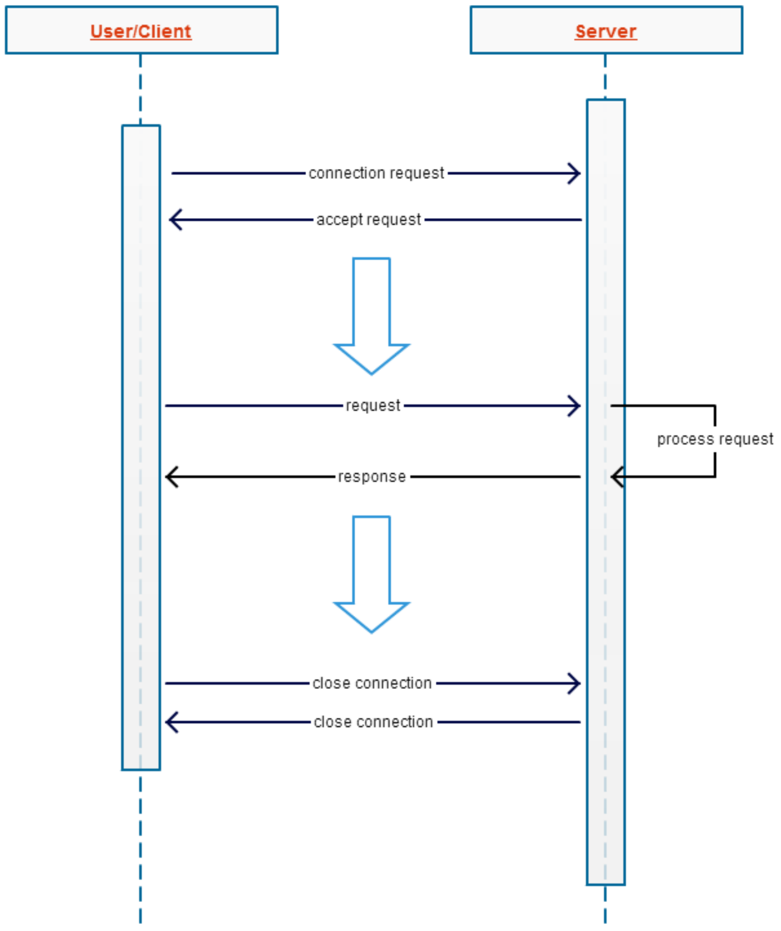

Figure 1: Epoll Echo Server Sequence Diagram

Given that the TCP echo server was able to handle upwards of 23000 concurrent connections, using the same design would help the port forwarder in performance. A weakness of the epoll design is that it treats all connections equally. It relies on a connection count to decide which thread to assign a connection. However, while it may not be more reliable, another measure to determine this could be average bytes sent per second. A thread handling large payloads will not be able to respond to smaller requests as quickly. Figure 2 shows the sequence diagram for the port forwarder, building off of the core connections found from the echo server.

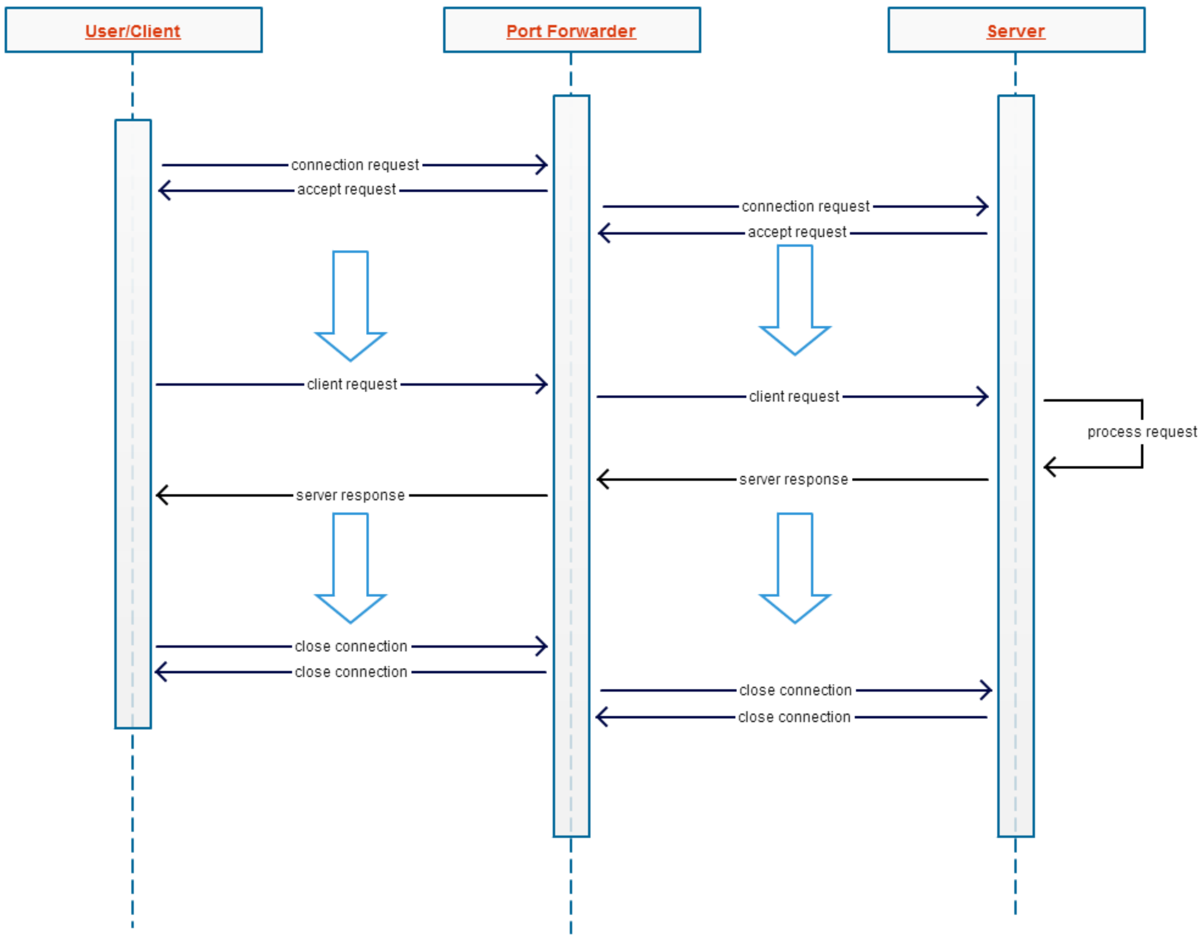

Figure 2: Port Forwarder Sequence Diagram

The main thread is responsible for creating child threads and outputting forward information to an output file. From the design decisions found in the epoll server, it was decided to run 8 child threads to handle requests and accepting new connections. Past research showed that having too many epoll worker threads did not benefit the application. Ideally, each worker thread would have its own core to operate on, but with threaded programming, this is not always the case. In addition to these child threads, another thread was created for accepting new connections. The accept thread allowed the program to accept new connections at a consistent speed, even if the worker threads are all busy processing requests. Having all 9 threads accepting new connections at the beginning of runtime allowed the application to pick up concurrent connections at a much faster rate. The worker threads sent connection details to the main thread through a pipe, which was constantly read and transferred to the output file. In addition, the accept thread had access to 8 pipes, one for each worker thread. The accept thread would decide which worker thread had the smallest workload and send new connections down that pipe. The worker thread would poll the pipe for new data along with its original duties. Refer to Figure 3 for a state machine diagram of the port forwarder application and its multi-threaded design.

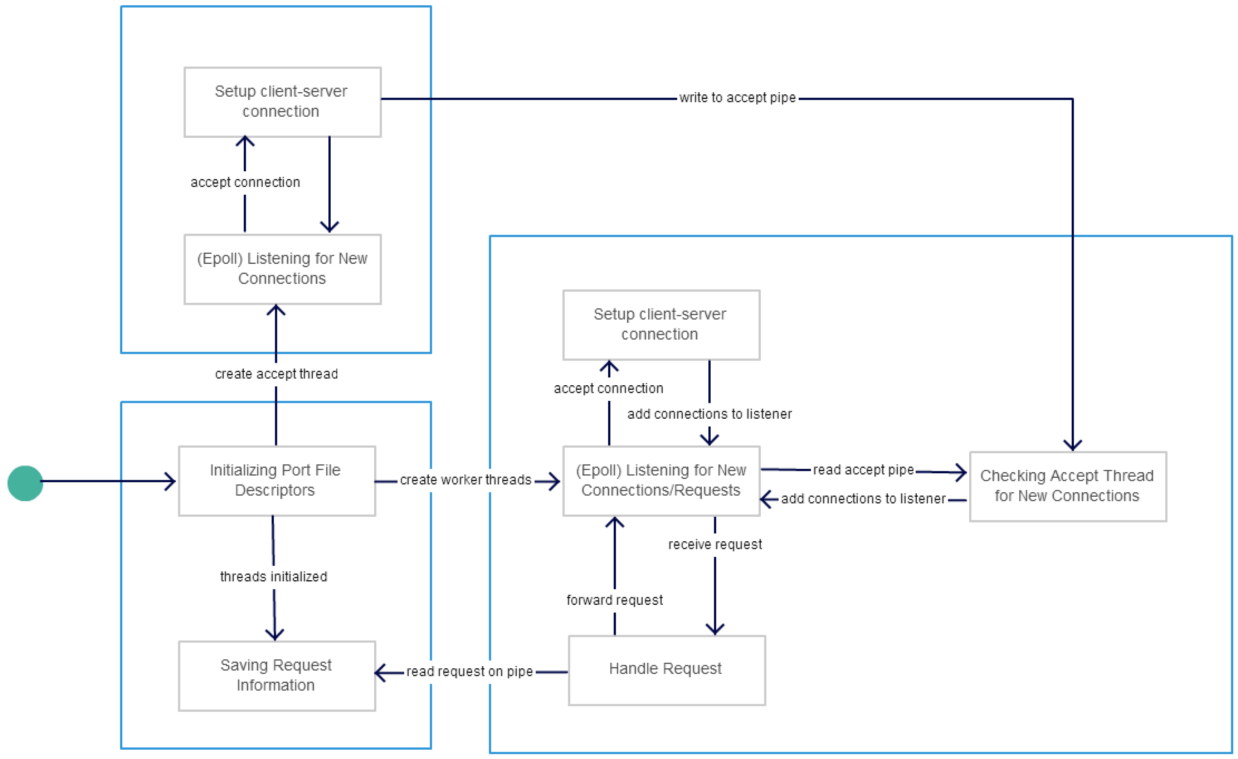
Setup client-server connection
write to accept pipe
accept connection
(Epoll) Listening for New Connections
create accept thread
Setup client-server connection
accept connection
add connections to listener
Initializing Port File Descriptors
create worker threads
(Epoll) Listening for New Connections/Requests
read accept pipe
add connections to listener
Checking Accept Thread for New Connections
receive request
threads initialized
forward request
Saving Request Information
read request on pipe
Handle Request

Figure 3: Port Forwarder State Machine Diagram

The port forwarder was implemented with a configuration file to read at start-up. This configuration file contains the expected ports clients will connect to and the destination server and port the data will be forwarded to. The accept thread and worker threads added the file descriptors for every forwarded port in the port forward table configuration file to its epoll set. Unlike the epoll server, the port forwarder needed to be able to start connections incoming from any of the listed ports. Each connection was now composed of an accepted client connection and being connected-to a server. The worker threads would epoll both the client and server file descriptors to relay the received data to the other side. The system uses the unique connection’s file descriptor to determine where to send the data.