Steven McVicker

Joseph Goergen

TFTP Client Write-up

We installed the FindBugs plugin for java, and saw some minor issues with our code. FindBugs reported our first issue as class names should start with an uppercase letter. We fixed that with an easy name change. The next warning was that file may fail to close stream. We fixed this by putting a inputstream.close at the end of the method. Lastly, it gave us a warning about not having a default case for our switch statement. These were minor issues, however in some cases could cause issues.

During our project, we ran into many problems that we had to work through. Our first major issue was finding out what classes were available and which ones we were going to use. We solved that by going through java documentation online and importing what we needed as we use it, for example the Datagram packet requires an InetAddress class so we imported that as well. Our next big issue came with receiving an acknowledgement packet and their port numbers. After Re-reading the RFC a few times we caught on to the pattern of the port destination and source, as well as the java functions that allowed us to grab those port numbers. We found out that the source port of the packet is our destination port. Also, after we sent the first read/write request, we would continually get an error message back. This error message was because we misplaced the 0 before our mode in the request byte array. We fixed this by making sure we placed the 0 in the correct spot. Once we got passed the initial send and receive and ack, it worked until we had to decide what was going to be the last packet. We were using the Datagram packet object and we did a “.data” which gave us an array of bytes including the 0’s, giving us a full packet length. Later we found that the object had a “.length” that did not include the extra 0’s at the end. This allowed us to be able to say if it was less than 512 bits then it was the last packet. One of the biggest issues we ran into for when we were sending a file was with the block numbers. Since there were two independent bytes for one block number, we had to create a function that took in a byte array and returned a byte array where it increments the value by one and if the rightmost byte reaches its max, then it adds one to the leftmost byte and sets the rightmost byte to zero. When we had to write back to the file we kept getting unknown chars in our file. The input stream object that we were using counted the opcode and block number (the first 4 bytes) and we did not account for that. We solved this by using the “.read” function from the input stream object and have it offset by 4 every time it reads.