Omar Abdelwahab
10151446

CPSC 441 ASG 2 Octoput Tansfer Protocol

Compiling:

```
g++ udp_server -o udp_server
g++ udp_client -o udp_client
```

To run:

First run the server by running ./server in your terminal.
Then on the client's terminal run it using ./udp_client fileName

Design choices:

- I'm using a pull model. I found it easier to conceptualize. It is safer because the client knows exactly what information they are requesting. It also removed the need for a sequence check because there is only one request or recieve happening at any time. It also eliminated most of the need for ACKs because if the client didn't recieve anything it needed it just requested it again after a known amount of time.
- I built the octopus on the client side based on the size to put less work on the server for scalability. However the server can only handle one file transfer at a time.
- I used the built in timeout for recvfrom in sockets, so I didn't need to multi-thread for the timer that was shown in class. I chose that because it made my code simpler, more efficient, more reliable, faster. There were no downsides in this because its part of the sockets class in C.
- I used structures for my file version of a file, octoblock, octolegs. This helped make it more independent of the server only sending 3 numbers, the ID, start and end of each octoleg request. This reduces the CPU workload on the server making it more efficient and distributing it to the clients. It also emphasizes the pull model mentiond earlier.

UDP transport layer protocol:

- Lower overhead
- Faster
- Increased work on application layer to implement my own checks and balances.
- Depending on how impotant and delicate the information is I wouldn't use UDP for transfering important data.
- Generally speaking applications that transfer data over network need to focus on other aspects of what they do, therefore the devlopers wouldn't have much time to implement a reliable, efficient connection over UDP so it isn't worth attempting it when TCP exists.
- Scales better because there is less overhead, and the individual packet sizes are relatively small because of the octoput.
- No parameters are built in for the user to play around with because this is a single use system, but it can be altered (with a lot of work) to work for other systems, and the parameters would be built custom for each system.
- The code is portable as it implements everything in the application layer so it doesn't affect it in any way if the lower layers are connectionless or connection oriented. It also doesn't care what transport layer is being used (wif, ethernet, etc.).

Testing:

- I tested on a loopback network on my VM running Ubuntu.
- I ran all the test files provided including the 256kb file. They all succeeded.
- I tried running the client without a running server.
- I tried running the client first then the server.
- I tried shutting down the server part way through.
- I used cmp newFile oldFile to check that the two files are the same and it succeeded in all cases.

Known Issues:

- If the client shuts down partway through the server **might** get a memoryleak or segfault eventually.(not tested for time reasons).
- ACKs are there but don’t really have any real use.
- My report is too long sorry.