Flinders UNIVERSITY
College of Science & Engineering

COMP2881 Operating Systems, DE
COMP9881 Operating Systems GE
Semester 1, 2020

CN Workshop07: Working with IP/UDP

Due: (check submission time on FLO)

Lab 7 is comprised of 2 tasks. This workshop contributes 10% to your final grade. Following completion of the checkpoints, a copy of the appropriate, makefile, source code and/or output for each task should be compiled into a single text file and uploaded to the relevant submission box on FLO for moderation. Speak to your demonstrator if you are unsure how to do this. While discussion of programming tasks with fellow students in labs is welcome and encouraged, please be mindful of the University's **Academic Integrity** policy and refrain from simply copying another student's code (even if they agree!). You will gain a much greater understanding of the material if you take the time to work through your own solutions, and you will be expected to be able to explain your code before being awarded checkpoints. Remember, the demonstrators are there to help if you get stuck.

Objectives

Upon completing this lab, you should be able to:
- Demonstrate an understanding of Computer Network concepts; IP addresses, sockets, simplex vs duplex connections, network protocols
- Demonstrate you are able to develop programs in the UNIX environment in C
- Demonstrate you are able to articulate user program design choices in terms of; security, efficiency, performance, or other relevant requirements

Learning out comes;
- LO1: Understand computer network terminology and topologies, functions and architectures of computer networks
- LO2: Demonstrate the design and implementation of simple process and network level programs
- LO3: Demonstrate an understanding of the various layers which make up networks and how they are used to form complex, reliable and secure communication mediums
- LO4: Understand and be able to articulate the structure and function of the major types of networks in common use
- LO5: Understand aspects of the internet, including various associated protocols IP, UDP, TCP, IP addressing, header fields and the operation of servers / clients
- LO6: Appreciate the real-time nature of networked devices and communications and demonstrate an understanding of current and emerging methods for secure information transfer

Preparation

Read through the entire workshop before you begin.

You may need to use tcpdump and/or wireshark to inspect the packets in flight over the wire/air.

Down load the example IP/UDP source code to use as a starting point;
https://github.com/amineamanzou/UDP-TCP-File-Transfer

Compile the TCP client and server programs to make sure they work in your environment. As ports 0-1024 need root user access, select a port number above 1024, so a regular user can run a network service to listen. Check `/etc/services` for unused ports.

Download the CTAP source code an use the `ctap.h` file in your client program;
https://github.com/jhunt/ctap
Read through the `README.md` file and make note of functions available.

Explore using stat() to find the size of a file;
https://www.tutorialspoint.com/unix_system_calls/stat.htm

Read documentation as appropriate;
http://cnp3book.info.ucl.ac.be/2nd/cnp3bis.pdf
https://beej.us/guide/bgnet/pdf/bgnet_A4_2.pdf

Task 1

Modify the provided IP/TCP server code to provide a service like HTTP.

The purpose of this service is to provided a mechanism to move files over a data stream service. Document each step in your implementation, marks will be allocated to informative commentary on what the code is doing.

The server should;

- Use a port number provided by an argument to `getopt()`, check it is between 40000-59999
- Create a socket on the provided port number
- Listen on the socket
- Wait to accept incoming file requests
- Open the local file, obtain the file size from stat()
- Respond by sending back the valid file name, and the file size or
  - If file does not exist, print status of "`Failed`", close connection (see example below), skip to Sleep
- Wait for the client to respond with the valid file size
- Respond by sending the file contents to the client
- Wait for the client to respond with "OK" or "Failed"
- Print status (see example below)
- Close the connection to the client
- Sleep for one second
- Start the process again

This program should be run on the command line and listen on the localhost IP or the IP of the available network interface. The server should print out the details of the IP and port number when it starts.

Example;

```
$ tcpserver -p 45678
Listening on IP: 127.0.0.1 and port: 45678
Connection: "1", file: "test1.txt", size: "31337", status: "OK"
Connection: "2", file: "test2.txt", size: "0", status: "Failed"
```

Task 2

Modify the provided IP/TCP client code to provide a test program to test the above server.

Use two separate sub-directories, one for the server, another for the client.

Run each in their own terminals and respective directories so that it is obvious that the file is transferred.

While test are supposed to be self documenting, please use meaningful messages in your tests.

Create appropriate unit tests using CTAP to;

- Loop over a filename that does exist and a filename that does not exist in the directory of the server
  - TEST ok: Get the start time
  - TEST ok: Create a client connection on a port and IP address, stored in global variables
  - TEST diag: Display the filename to be requested
  - TEST ok: Send a request to the server
  - TEST ok: Wait for the response from the server
  - …
  - TEST ok: Check the response payload from the server matches filename an bytes are more than 0(zero)
    - TEST ok: Open the new file
    - Transfer the data,
      - Save the data to the file
    - TEST ok: stat() the size of the file in bytes confirm that the sizes match
  - …
  - TEST diag: Display response pay load
  - TEST ok: Get the finish time
  - TEST diag: Calculate and display time taken to complete the network connection
  - TEST diag: Calculate the bytes per second of the file sent over the network connection
  - TEST ok: Check that the time taken is less than one second

The goal here is to write a program that checks that the server sends the file correctly.

More tests may be required, feel free to add any steps you think are necessary.

It should be noted that your should test with files of a few kilobytes, a few tens of kilobytes and a file of at least a megabyte to demonstrate that the server and client can loop of the data is a small buffer, accurately dealing with any type of file.

For more information on see;

https://beej.us/guide/bgnet/pdf/bgnet_A4_2.pdf