Lab 5 Streaming Fileserver
May 1st, 2014
Dan Kass
CE 4960 (Networking II)

Benchmarking

File	UDP	TCP
OSI.pdf	4.772s	0.020s
Attackgoat.gif	6.059s	0.022s
Download.jpg	0.097s	0.004s
Highscoreos.zip	0.030s	0.004s
listing	0.011s	0.003s

Source Code

```
/*
 ============================================================================
 Name        : lab5.c
 Author      : Dan Kass
 Version     :
 Copyright   :
 Description : A simple file server using TCP
 ============================================================================
 */

// Simple TCP echo server
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <dirent.h>

// Max message to echo
#define MAX_MESSAGE     1000

/* server main routine */

int main(int argc, char** argv) {

      // locals
      unsigned short port = 22222; // default port
      int sock; // socket descriptor

      // Was help requested?  Print usage statement
      if (argc > 1 && ((!strcmp(argv[1], "-?")) || (!strcmp(argv[1], "-h")))) {
            printf(
                        "\nUsage: tcpechoserver [-p port] port is the requested \
port that the server monitors.  If no port is provided, the server \
listens on port 22222.\n\n");
           exit(0);
     }

     // get the port from ARGV
     if (argc > 1 && !strcmp(argv[1], "-p")) {
           if (sscanf(argv[2], "%hu", &port) != 1) {
                 perror("Error parsing port option");
                 exit(0);
           }
     }

     // ready to go
     printf("tcp fileserver configuring on port: %d\n", port);

     // for TCP, we want IP protocol domain (PF_INET)
     // and TCP transport type (SOCK_STREAM)
     // no alternate protocol - 0, since we have already specified IP

     if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
           perror("Error on socket creation");
           exit(1);
     }

     // establish address - this is the server and will
     // only be listening on the specified port
     struct sockaddr_in sock_address;

     // address family is AF_INET
     // our IP address is INADDR_ANY (any of our IP addresses)
     // the port number is per default or option above

     sock_address.sin_family = AF_INET;
     sock_address.sin_addr.s_addr = htonl(INADDR_ANY);
     sock_address.sin_port = htons(port);

     // we must now bind the socket descriptor to the address info
     if (bind(sock, (struct sockaddr *) &sock_address, sizeof(sock_address))
                 < 0) {
           perror("Problem binding");
           exit(-1);
     }

     // extra step to TCP - listen on the port for a connection
     // willing to queue 5 connection requests
     if (listen(sock, 5) < 0) {
           perror("Error calling listen()");
           exit(-1);
     }

     // go into forever loop and echo whatever message is received
     // to console and back to source
     char* buffer = calloc(MAX_MESSAGE, sizeof(char));
     char* list = calloc(MAX_MESSAGE, sizeof(char));
     char* filename = calloc(MAX_MESSAGE + 20, sizeof(char));
     char* dir = "/home/kassd/uploads/";
     char insert = 0;
```

```
int bytes_read;
int connection;
FILE *fp;

while (1) {

      // hang in accept and wait for connection
      printf("====Waiting====\n");
      if ((connection = accept(sock, NULL, NULL)) < 0) {
            perror("Error calling accept");
            exit(-1);
      }

      // ready to r/w - another loop - it will be broken when
      // the connection is closed
      while (1) {
            // reset buffer counter
            insert = 0;

            buffer[0] = '\0'; // guarantee a null here to break out on a
            list[0] = '\0';
            filename[0] = '\0';

            // copy null-terminated string into buffer
            // go until we get a null terminator or and endline

            while (1) {
                  if ((bytes_read = read(connection, &buffer[insert], 1)) < 0) {
                        perror("Error calling read");
                        exit(-1);
                  }

                  if (bytes_read == 0)
                        break;

                  if (buffer[insert] == '\0') {
                        break;
                  }

                  insert++;
            }

            //check for file listing
            if (strncmp(buffer, ".", 1) == 0) {

                  //get list of files from directory

                  DIR *dp;
                  struct dirent *ep;
                  dp = opendir(dir);

                  if (dp != NULL) {
                        int j = 0;
                        while (ep = readdir(dp)) {
                              int i = 0;
                              //checking for the .. and . in the
                              //directory and ignoring them
                              if (!((strncmp(ep->d_name, "..", 2) == 0)
                    || (strncmp(ep->d_name, ".", 1) == 0))) {
                while (ep->d_name[i] != '\0') {
                    list[j++] = ep->d_name[i++];
                }
                list[j++] = '\n';
            }
        }
        list[j - 1] = '\0';
        (void) closedir(dp);
    }

    if ((write(connection, list, strlen(list))) < 0) {
        perror("Error sending listing");
        exit(-1);
    } else {
        printf("====Listing Requested====\n");
    }
    close(connection);
    break; // break the inner while loop
}

if (bytes_read == 0) { // socket closed
    printf("====Client Disconnected====\n");
    close(connection);
    break; // break the inner while loop
}

// see if client wants us to disconnect
if (strncmp(buffer, "quit", 4) == 0) {
    printf("====Server Disconnecting====\n");

    close(connection);
    break; // break the inner while loop
}

//ok so it must be a file name!
//read the file name put it into the buffer and
printf("====%s Requested====\n", buffer);
FILE *fp;
strcpy(filename, dir);
strcat(filename, buffer);
fp = fopen(filename, "r");
if (fp == NULL) {
    printf("File Doesn't Exists\n");

    close(connection);
    break; // break the inner while loop
}
int sizesent = 0;
while ((sizesent = fread(list, 1, MAX_MESSAGE, fp)) != 0) {

    // send it back to client
    if ((send(connection, list, sizesent, 0)) < 0) {
        perror("Error sending file");
        exit(-1);
    } else {
        printf("Packet Sent\n");
    }
```

```
                  }
                  //disconnect
                  fclose(fp);
                  close(connection);
                  break; // break the inner while loop

            } // end of accept inner-while

      } // end of outer loop

      // will never get here
      return (0);
}
```