```
README
All src files (*.c) should be in src/ folder
There should be a obj/ folder to storage object files (*.o)

Test Case:
================== Requirement ==================
$ ls
Makefile  tcpFileClient  tcpPPRServer  udpFileClient  udpPPRServer
obj       tcpiServer     tcpPTServer   udpiServer     udpPTServer
src       tcpPFServer    tcpTPRServer  udpPFServer    udpTPRServer

================== Test TCP ==================
================== Test Iterative Server ==================
$ ./tcpiServer
open file: /bin/which
start sending...
close file
done sending

$ ./tcpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Process Per Request Server ==================
$ ./tcpPPRServer
open file: /bin/which
start sending...
close file
done sending

$ ./tcpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Thread Per Request Server ==================
$ ./tcpTPRServer
open file: /bin/which
start sending...
close file
done sending

$ ./tcpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Pre-forked Server ==================
$ ./tcpPFServer 10
open file: /bin/which
start sending...
close file
done sending

$ ./tcpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Pre-threaded Server ==================
$ ./tcpPTServer 10
open file: /bin/which
start sending...
close file
done sending

$ ./tcpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== End Test TCP ==================

================== Test UDP ==================
================== Test Iterative Server ==================
$ ./udpiServer
open file: /bin/which
start sending...
close file
done sending

$ ./udpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Process Per Request Server ==================
$ ./udpPPRServer
open file: /bin/which
start sending...
close file
done sending

$ ./udpFileClient
File name: /bin/which
createFile: which
start get file ...
```

```
done
$ diff which /bin/which
================== Test Thread Per Request Server ==================
$ ./udpTPRServer
open file: /bin/which
start sending...
close file
done sending

$ ./udpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Pre-forked Server ==================
$ ./udpPFServer 10
open file: /bin/which
start sending...
close file
done sending

$ ./udpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== Test Pre-threaded Server ==================
$ ./udpPTServer 10
open file: /bin/which
start sending...
close file
done sending

$ ./udpFileClient
File name: /bin/which
createFile: which
start get file ...
done
$ diff which /bin/which
================== End Test UDP ==================


================== source code ==================
++++++++++++++++++++++++Makefile++++++++++++++++++++++++
# Make file
# all source code should be in src folder
# all objects will be place in obj folder
# link necessary objects to create exe program

#================== Source Files ==================
ERREXIT_SRC = src/errexit.c
SERVERS_SRC = src/passivesock.c src/udpiServer.c src/udpTPRServer.c
CLIENTS_SRC = src/connectsock.c src/udpFileClient.c
#====================================================

#================== Object Files ===================
SERVER_OBJ = obj/errexit.o obj/passivesock.o
CLIENT_OBJ = obj/errexit.o obj/connectsock.o
UDP_OBJ    = obj/udpFileServer.o
TCP_OBJ    = obj/tcpFileServer.o
#====================================================

#================ Executable Files =================
UDP = udpFileClient udpiServer udpTPRServer udpPTServer udpPPRServer udpPFServer
TCP = tcpFileClient tcpiServer tcpTPRServer tcpPTServer tcpPPRServer tcpPFServer
PROGRAM = $(UDP) $(TCP)
#====================================================

all: $(PROGRAM)

udp: $(UDP)

tcp: $(TCP)

obj/%.o: src/%.c
	@gcc -Wall -c $< -o $@

udpFileClient: obj/udpFileClient.o $(CLIENT_OBJ)
	gcc -Wall -o udpFileClient obj/udpFileClient.o $(CLIENT_OBJ)

udpiServer: obj/udpiServer.o $(SERVER_OBJ) $(UDP_OBJ)
	gcc -Wall -o udpiServer obj/udpiServer.o $(SERVER_OBJ) $(UDP_OBJ)

udpTPRServer: obj/udpTPRServer.o $(SERVER_OBJ) $(UDP_OBJ)
	gcc -Wall -pthread -o udpTPRServer obj/udpTPRServer.o $(SERVER_OBJ)
$(UDP_OBJ)

udpPTServer: obj/udpPTServer.o $(SERVER_OBJ) $(UDP_OBJ)
	gcc -Wall -pthread -o udpPTServer obj/udpPTServer.o $(SERVER_OBJ) $(UDP_OBJ)

udpPPRServer: obj/udpPPRServer.o $(SERVER_OBJ) $(UDP_OBJ)
	gcc -Wall -o udpPPRServer obj/udpPPRServer.o $(SERVER_OBJ) $(UDP_OBJ)

udpPFServer: obj/udpPFServer.o $(SERVER_OBJ) $(UDP_OBJ)
	gcc -Wall -o udpPFServer obj/udpPFServer.o $(SERVER_OBJ) $(UDP_OBJ)

tcpFileClient: obj/tcpFileClient.o $(CLIENT_OBJ)
	gcc -Wall -o tcpFileClient obj/tcpFileClient.o $(CLIENT_OBJ)

tcpiServer: obj/tcpiServer.o $(SERVER_OBJ) $(TCP_OBJ)
	gcc -Wall -o tcpiServer obj/tcpiServer.o $(SERVER_OBJ) $(TCP_OBJ)
```

```
tcpTPRServer: obj/tcpTPRServer.o $(SERVER_OBJ) $(TCP_OBJ)
	gcc -Wall -pthread -o tcpTPRServer obj/tcpTPRServer.o $(SERVER_OBJ)
$(TCP_OBJ)

tcpPTServer: obj/tcpPTServer.o $(SERVER_OBJ) $(TCP_OBJ)
	gcc -Wall -pthread -o tcpPTServer obj/tcpPTServer.o $(SERVER_OBJ) $(TCP_OBJ)

tcpPPRServer: obj/tcpPPRServer.o $(SERVER_OBJ) $(TCP_OBJ)
	gcc -Wall -o tcpPPRServer obj/tcpPPRServer.o $(SERVER_OBJ) $(TCP_OBJ)

tcpPFServer: obj/tcpPFServer.o $(SERVER_OBJ) $(TCP_OBJ)
	gcc -Wall -o tcpPFServer obj/tcpPFServer.o $(SERVER_OBJ) $(TCP_OBJ)

clean:
	@rm -v obj/*.o $(PROGRAM)
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++connectsock.c++++++++++++++++++++++++
/* connectsock.c - connectsock */
#include "connectsock.h"

/*------------------------------------------------------------------------
 * connectsock - allocate & connect a socket using TCP or UDP
 *------------------------------------------------------------------------
 */

/**
 * Arguments:
 * node - name of host to which connection is desired
 * service - service associated with the desired port
 * transport - name of transport protocol to use ("tcp" or "udp")
 * @author: Aaron Lam
 */
int connectsock(const char *node, const char *service, const char *transport ) {
    int sockfd; // socket descriptor
    int errcode; // error code return by getaddrinfo
    struct addrinfo hints; // address info hints
    struct addrinfo *res; // address info linked list that has the result

    memset(&hints, 0, sizeof(hints));

    hints.ai_family = AF_INET; // set to use IPv4 only

    // set socket type
    if( (strcmp(transport, "udp")) == 0) {
        hints.ai_socktype = SOCK_DGRAM;
    } else {
        hints.ai_socktype = SOCK_STREAM;
    }

    // get address info; if error, print error then exit
    if ((errcode = getaddrinfo(node, service, &hints, &res)) != 0) {
        errexit("getaddrinfo(): %s\n", gai_strerror(errcode));
    }

    // loop through all the results and connect to the first valid address
    struct addrinfo *it; // res linkedlist iterator
    for(it = res; it != NULL; it = it->ai_next) {
        // try socket
        // if fail, try next address
        if ((sockfd = socket(it->ai_family, it->ai_socktype, it->ai_protocol)) ==
-1) {
            continue;
        }
        // try connect
        // if fail, close the current socket descriptor then try next address
        if ( (connect(sockfd, it->ai_addr, it->ai_addrlen)) == -1) {
            close(sockfd);
            continue;
        }
        break; // connected successfully
    }
    // free res, not needed anymore
    freeaddrinfo(res);

    if (it == NULL) {
        // looped off the end of the list with no connection
        errexit("connectsock fail!\n");
    }

    return sockfd;
}
++++++++++++++++++++++++END+++++++++++++++++++++++
++++++++++++++++++++++++connectsock.h++++++++++++++++++++++++
/**
 * @author: Aaron Lam
 */
#ifndef CONNECTSOCK_H_
#define CONNECTSOCK_H_

#include "errexit.h"

#include <errno.h>

#include <sys/types.h>
#include <sys/socket.h>

#include <netinet/in.h>
#include <arpa/inet.h>

#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
```

```
#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif /* INADDR_NONE */

typedef unsigned short u_short;

int connectsock (const char *host, const char *service, const char *transport);

#endif
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++errexit.c++++++++++++++++++++++++
/* errexit.c - errexit */
#include "errexit.h"

/*------------------------------------------------------------------------
* errexit - print an error message and exit
*------------------------------------------------------------------------
*/
/*VARARGS1*/
int
errexit(const char *format, ...)
{
  va_list args;
  va_start(args, format);
  vfprintf(stderr, format, args);
  va_end(args);
  exit(1);
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++errexit.h++++++++++++++++++++++++
#ifndef ERREXIT_H_
#define ERREXIT_H_

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

int
errexit(const char *format, ...);

#endif
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++passivesock.c++++++++++++++++++++++++
/* passivesock.c - passivesock */

#include "passivesock.h"

typedef unsigned short u_short;
```

```
u_short portbase = 0; /* port base, for non-root servers */

/*------------------------------------------------------------------------
* passivesock - allocate & bind a server socket using TCP or UDP
*------------------------------------------------------------------------
*/

int
passivesock(const char *service, const char *transport, int qlen)
/*
* Arguments:
* service - service associated with the desired port
* transport - transport protocol to use ("tcp" or "udp")
* qlen - maximum server request queue length
*/
{
    struct servent *pse; /* pointer to service information entry */
    struct protoent *ppe; /* pointer to protocol information entry*/
    struct sockaddr_in sin; /* an Internet endpoint address */
    int s, type; /* socket descriptor and socket type */
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;

    /* Map service name to port number */
    if ( (pse = getservbyname(service, transport)) ) {
        sin.sin_port = htons(ntohs((u_short)pse->s_port) + portbase);
    }
    else if ( (sin.sin_port = htons((u_short)atoi(service))) == 0 ) {
        errexit("can't get \"%s\" service entry\n", service);
    }

    /* Map protocol name to protocol number */
    if ( (ppe = getprotobyname(transport)) == 0) {
        errexit("can't get \"%s\" protocol entry\n", transport);
    }

    /* Use protocol to choose a socket type */
    if (strcmp(transport, "udp") == 0) {
        type = SOCK_DGRAM;
    }
    else {
        type = SOCK_STREAM;
    }

    /* Allocate a socket */
    s = socket(PF_INET, type, ppe->p_proto);
    if (s < 0) {
        errexit("can't create socket: %s\n", strerror(errno));
    }

    /* Bind the socket */
    if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("can't bind to %s port: %s\n", service, strerror(errno));
    }
    if (type == SOCK_STREAM && listen(s, qlen) < 0) {
        errexit("can't listen on %s port: %s\n", service, strerror(errno));
    }

    return s;
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++passivesock.h++++++++++++++++++++++++
#ifndef PASSIVESOCK_H
#define PASSIVESOCK_H

#include "errexit.h"

#include <errno.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>


int passivesock(const char *service, const char *transport, int qlen);

#endif
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpFileClient.c++++++++++++++++++++++++
/*********************************************************************/
/* Author: Aaron Lam                                                 */
/* Description: TCP File Client, connect to TCP file server          */
/*     get file path from user, then send request to server          */
/*     get file name return from server, create file with file name  */
/*     get file content from server and write to local file          */
/*********************************************************************/


/* tcpFileClient.c - main */

#include "errexit.h"
#include "connectsock.h"

#include <unistd.h>
#include <fcntl.h>
#include <string.h>

/**
 * sendFileRequest: get filepath from user, send filepath to server using socket
 * Arguements:
```

```
 * sock     - socket descripter, current open socket
 * buf      - file path, using default BUFSIZ
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
int sendFileRequest (int sock, char *buf) {
    memset(buf, 0, BUFSIZ);
    printf("File name: ");
    fgets(buf, BUFSIZ, stdin);
    buf[strlen(buf) - 1] = '\0';

    if (sendto(sock, buf, strlen(buf), 0, NULL, 0) < 0) {
        errexit("sendto() error: %s\n", strerror(errno));
    }
    return 0;
}

/**
 * createFile: get file name from server, create a file with that name
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - storage filename recv from server, using default BUFSIZ
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
int createFile (int sock, char *buf) {
    memset(buf, 0, BUFSIZ);
    if ((recvfrom(sock, buf, BUFSIZ, 0, NULL, 0)) > 0) {
        printf("createFile: %s\n", buf);
        mode_t mode = 0666;
        int fd = open(buf, O_WRONLY | O_CREAT, mode);
        if ( fd == -1 ) {
            errexit("open() error: %s\n", strerror(errno));
        }
        return fd;
    }
    return -1;
}

/**
 * getFile: get file content from server, then write content to file in fd
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - storage filename recv from server, using default BUFSIZ
 *          - use as buf for message get from server
 * fd       - file descriptor, current open file
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
```

```
int getFile (int sock, char *buf, int fd) {
    printf("start get file ...\n");
    if (fd == -1) return -1;
    int n;
    while ( (n = recvfrom(sock, buf, BUFSIZ, 0, NULL, 0)) > 0) {
        if (write (fd, buf, n) < 0 ) {
            errexit("write() error: %s\n", strerror(errno));
        }
    }
    if (n == -1) {
        errexit("recvfrom() error: %s\n", strerror(errno));
    }
    close(fd);
    printf("done\n");
    return 0;
}

void run_client(int sock) {
    char buf[BUFSIZ];
    int fd;

    if (sendFileRequest(sock, buf) == -1) {
        errexit("sendFileRequest() error: %s\n", strerror(errno));
    }

    if ((fd = createFile(sock, buf)) == -1){
        errexit("createFile() error: %s\n", strerror(errno));
    }

    if (getFile(sock, buf, fd) == -1) {
        errexit("getFile() error: %s\n", strerror(errno));
    }
}

/*------------------------------------------------------------------------
 * main - TCP File Client for TCP File Service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    char *host = "localhost"; /* host to use if none supplied */
    char *service = "9000"; /* default service name */

    switch (argc) {
        case 1:
        host = "localhost";
        break;
        case 3:
        service = argv[2];
        /* FALL THROUGH */
        case 2:
        host = argv[1];
        break;
        default:
        fprintf(stderr, "usage: tcpFileClient [host [port]]\n");
        return 1;
    }

    int sock = connectsock(host, service, "tcp");

    run_client(sock);

    return 0;
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpFileServer.c++++++++++++++++++++++++
#include "tcpFileServer.h"

/**
 * @author: Aaron Lam
 */

/**
 * getFilePath: get file path from client connected to current socket.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - source address, will be filled by this function
 *          - src_addr need to be saved for later use
 * socklen  - socket length, size of src_addr
 * Return:
 * the requested file path from client is save buf for later use
 * return nothing. Report error code (errno) if recvfrom() fail
 */
void getFilePath (int sock, char *buf, struct sockaddr_in *src_addr, socklen_t
*socklen) {
    int n = recvfrom(sock, buf, BUFSIZ, 0, (struct sockaddr *) src_addr,
socklen);
    if (n < 0) {
        printf("recvfrom() error: %s\n", strerror(errno));
    }
    buf[n] = '\0';
    return;
}

/**
 * sendFileName: send file name back to client, act as connection check.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - server source address, pass in as value
 * socklen  - socket length, size of src_addr
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 */
```

```
void sendFileName (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen) {
    char filename[BUFSIZ];
    memset(filename, 0, BUFSIZ);
    strcpy(filename, basename(buf));
    filename[strlen(filename)] = '\0';

    int count = sendto(sock, filename, strlen(filename)+1, 0, (struct sockaddr
*)&src_addr, socklen);
    if (count < 0) {
        printf("sendto() error: %s\n", strerror(errno));
    }
}

/**
 * sendFile: open the requested file, then send file back to client.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - server source address, pass in as value
 * socklen  - socket length, size of src_addr
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 */
void sendFile (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen) {
    printf("open file: %s\n", buf);
    int fd = open(buf, O_RDONLY); //open file identify by file path
    if ( fd == -1 ) {
        sendto(sock, NULL, 0, 0, (struct sockaddr *)&src_addr, socklen);
        printf("error openning file: '%s' '%s'\n", buf, strerror(errno));
    }

    printf("start sending...\n");
    int n;
    while ( (n = read(fd, buf, BUFSIZ)) > 0 ) {
        sendto(sock, buf, n, 0, (struct sockaddr *)&src_addr, socklen);
    }
    close(fd);
    printf("close file\n");

    close(sock);
    printf("done sending\n");
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpFileServer.h++++++++++++++++++++++++
#ifndef TCP_FILE_SERVER_H
#define TCP_FILE_SERVER_H

#include "errexit.h"
#include "passivesock.h"
```

```
#include <unistd.h>
#include <fcntl.h>
#include <libgen.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>

void getFilePath (int sock, char *buf, struct sockaddr_in *src_addr, socklen_t
*socklen);

void sendFileName (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen);

void sendFile (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen);

#endif
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpiServer.c++++++++++++++++++++++++
/* UDP_iterativeServer.c - main */

#include "errexit.h"
#include "passivesock.h"
#include "tcpFileServer.h"

#define BACKLOG 10

/**
 * run_server: do server work - get file, and send file
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 */
void run_server (int server_sock) {
    int sock; /* client socket */
    struct sockaddr_in src_addr; /* the from address of a client */
    socklen_t socklen = sizeof(src_addr);

    char buf[BUFSIZ]; /* message buffer; use default stdio BUFSIZ */

    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for debug */
        memset (buf, 0, BUFSIZ);
        memset (&src_addr, 0, sizeof(src_addr));

        sock = accept(server_sock, (struct sockaddr *)&src_addr, &socklen);

        getFilePath(sock, buf, &src_addr, &socklen);
        sendFileName(sock, buf, src_addr, socklen);

        sendFile(sock, buf, src_addr, socklen);

        close(sock);
    }
}

/*------------------------------------------------------------------------
 * main - Iterative UDP server for file transfer service
 *------------------------------------------------------------------------
 */
int
main(int argc, char *argv[])
{
    int sock; /* server socket */
    char *service = "9000"; /* service name or port number */

    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: tcpiServer [port]\n");
    }

    sock = passivesock(service, "tcp", BACKLOG);

    run_server(sock);

    close(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpPFServer.c++++++++++++++++++++++++
/*********************************************************************/
/* Author: Aaron Lam                                                 */
/* Description: UDP file server, prefork server                      */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
/*********************************************************************/

/* udpTPRServer.c - main */

#include <pthread.h>

#include "errexit.h"
#include "passivesock.h"
#include "tcpFileServer.h"
```

```
int CHILD_COUNT = 5;

static int main_pid;

/* fork_child: make sure only main can fork
 * Arguments: nothing
 * Return:
 * return child pid or exit if already in child process
 * @author: Aaron Lam
 */
int fork_child()
{
    if(getpid() == main_pid)
        return fork();
    else
        exit(0);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 * @author: Aaron Lam
 */
void run_server (int server_sock) {
    int sock; /* client socket */
    main_pid = getpid();
    int count;

    for(count = 0; count < CHILD_COUNT; ++count) {
        int pid = fork_child();
        if(pid < 0) {
            printf("fork() error: %s", strerror(errno));
        }
        else if (pid == 0) {
            /* message buffer; use default stdio BUFSIZ */
            char buf[BUFSIZ];

            /* the from address of a client */
            struct sockaddr_in src_addr;
            /* socket length */
            socklen_t socklen = sizeof(src_addr);

            int run_flag = 1;
            while (run_flag) {
                // run_flag = 0; /* for mem leak check */

                /* zero out variable before use */
                memset(buf, 0, BUFSIZ);
                memset(&src_addr, 0, sizeof(src_addr));

                sock = accept(server_sock, (struct sockaddr *)&src_addr,
&socklen);

                /* do server work */
                getFilePath(sock, buf, &src_addr, &socklen);
                sendFileName(sock, buf, src_addr, socklen);
                sendFile(sock, buf, src_addr, socklen);
            }
            exit(0);
        }
    }

    wait(NULL);
}

/*---------------------------------------------------------------------------
 * main - UDP Prefork Server for file transfer service
 *---------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        CHILD_COUNT = atoi(argv[1]);
        break;
        case 3:
        service = argv[1];
        CHILD_COUNT = atoi(argv[2]);
        default:
        errexit("usage: tcpPFServer [port] [child_num]\n");
    }

    /* get server socket */
    sock = passivesock(service, "tcp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++tcpPPRServer.c+++++++++++++++++++++++++
/***************************************************************/
```

```
/* Author: Aaron Lam                                                 */
/* Description: UDP file server, process per request                 */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
/*********************************************************************/

/* udpTPRServer.c - main */

#include <pthread.h>
#include <sys/wait.h>

#include "errexit.h"
#include "passivesock.h"
#include "tcpFileServer.h"

static int main_pid;

/* fork_child: make sure only main can fork
 * Arguments: nothing
 * Return:
 * return child pid or exit if already in child process
 * @author: Aaron Lam
 */
int fork_child()
{
    if(getpid() == main_pid)
        return fork();
    else
        exit(0);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 */
void run_server (int server_sock) {
    int sock; /* client socket */
    main_pid = getpid();

    /* message buffer; use default stdio BUFSIZ */
    char buf[BUFSIZ];

    /* the from address of a client */
    struct sockaddr_in src_addr;

    socklen_t socklen = sizeof(src_addr);
```

```c
    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for mem leak check */

        /* zero out the variable before use */
        memset(buf, 0, BUFSIZ);
        memset(&src_addr, 0, sizeof(src_addr));

        sock = accept(server_sock, (struct sockaddr *)&src_addr, &socklen);

        int pid = fork_child();
        if(pid < 0) {
            printf("fork() error: %s", strerror(errno));
        }
        else if (pid == 0) {
            getFilePath(sock, buf, &src_addr, &socklen);
            sendFileName(sock, buf, src_addr, socklen);
            sendFile(sock, buf, src_addr, socklen);
            close(sock);
            exit(0);
        } else {
            close(sock);
        }
    }
    wait(NULL);
}

/*------------------------------------------------------------------------
 * main - UDP Thread Per Request Server for file transfer service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: udpPPTServer [port]\n");
    }

    /* get server socket */
    sock = passivesock(service, "tcp", 0);

    /* run server */
    run_server(sock);

    close(sock);

}
++++++++++++++++++++++END+++++++++++++++++++++++
++++++++++++++++++++++tcpPTServer.c+++++++++++++++++++++++
/*******************************************************************/
/* Author: Aaron Lam                                               */
/* Description: UDP file server, pre-threaded (thread pool) server   */
/*  get file name from client, use a thread pool to handle requests  */
/*  the server process is keep alive in a forever while loop         */
/*******************************************************************/

/* udpPTServer.c - main */

#include <pthread.h>

#include "errexit.h"
#include "passivesock.h"
#include "tcpFileServer.h"

int THREAD_COUNT = 5;

pthread_mutex_t mutex;

/**
 * handle_request: handle client request, call by pthread_create
 * Arguements:
 * input: socket that created in main thread, need to cast to int before use
 * Return:
 * return nothing.
 * get file path from client, then send file name and file content back to client
 * @author: Aaron Lam
 */
void *handle_request (void *input) {
    /* server socket descriptor */
    long *sock_pt = (long*) input;
    int server_sock = *sock_pt;
    int sock; /* client socket */

    /* message buffer; use default stdio BUFSIZ */
    char buf[BUFSIZ];

    /* the from address of a client */
    struct sockaddr_in src_addr;

    socklen_t socklen = sizeof(src_addr);

    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for mem leak check */
        /* zero out variable before use */
        memset(buf, 0, BUFSIZ);
        memset(&src_addr, 0, socklen);

        pthread_mutex_lock(&mutex);
        sock = accept(server_sock, (struct sockaddr *)&src_addr, &socklen);
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        getFilePath(sock, buf, &src_addr, &socklen);
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        sendFileName(sock, buf, src_addr, socklen);
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        sendFile(sock, buf, src_addr, socklen);
        pthread_mutex_unlock(&mutex);

    }
    pthread_exit(NULL);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 * @author: Aaron Lam
 */
void run_server (int sock) {

    pthread_t thread_ids[THREAD_COUNT];

    pthread_mutex_init(&mutex, NULL);
    pthread_attr_t attr;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    int i;
    for(i = 0; i < THREAD_COUNT; ++i) {
        pthread_create(&thread_ids[i], &attr, handle_request, (void *) &sock);
    }

    pthread_attr_destroy(&attr);
    for(i = 0; i < THREAD_COUNT; ++i) {
        pthread_join(thread_ids[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    pthread_exit(NULL);
}

/*------------------------------------------------------------------------
 * main - UDP Pre-Thread Server for file transfer service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        /* defaut case: port 9000, thread count 5 */
        case 1:
        break;
        case 2:
        THREAD_COUNT = atoi(argv[1]);
        break;
        case 3:
        service = argv[1];
        THREAD_COUNT = atoi(argv[2]);
        break;
        default:
        errexit("usage: tcpPTServer [port] [thread_count]\n");
    }

    /* get server socket */
    sock = passivesock(service, "tcp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END+++++++++++++++++++++++++
++++++++++++++++++++++++tcpTPRServer.c++++++++++++++++++++++++
/*********************************************************************/
/* Author: Aaron Lam                                                 */
/* Description: UDP file server, thread per request                  */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
/*********************************************************************/

/* udpTPRServer.c - main */

#include <pthread.h>
```

```c
#include "errexit.h"
#include "passivesock.h"
#include "tcpFileServer.h"

/**
 * handle_request: handle client request, call by pthread_create
 * Arguements:
 * input: storage ThreadData, need to be covert to ThreadData before use
 * Return:
 * return nothing. send file name and file content back to client
 * @author: Aaron Lam
 */
void *handle_request (void *input) {
    /* client socket */
    long *sock_pt = (long*) input;
    int sock = *sock_pt;

    /* message buffer; use default stdio BUFSIZ */
    char buf[BUFSIZ];
    memset(buf, 0, BUFSIZ);

    struct sockaddr_in src_addr;

    getFilePath(sock, buf, NULL, 0);
    sendFileName(sock, buf, src_addr, 0);
    sendFile(sock, buf, src_addr, 0);

    close(sock);

    pthread_exit(NULL);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error message (errno) if error
 * @author: Aaron Lam
 */
void run_server (int server_sock) {
    int sock; /* client socket */
    /* thread id */
    pthread_t thread;
    /* thread attribule */
    pthread_attr_t attr;

    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    /* set detach state so main don't have to call pthread_join */
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    /* while loop to keep server alive
     * variables needed for a thread are created in while
     * no share variables between threads except socket
     */
    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for mem leak check */


        /* the from address of a client */
        struct sockaddr_in src_addr;
        /* zero out the src_addr so it can be filled later */
        memset (&src_addr, 0, sizeof(src_addr));

        socklen_t socklen = sizeof(src_addr);

        sock = accept(server_sock, (struct sockaddr *)&src_addr, &socklen);

        /* create thread to handle request */
        if (pthread_create(&thread, &attr, handle_request, (void*) &sock) < 0) {
            printf("pthread_create(): %s\n", strerror(errno));
        }
    }

    /* destroy attribute to void memory leak */
    pthread_attr_destroy(&attr);

    /* main thread exit */
    pthread_exit(NULL);
}

/*------------------------------------------------------------------------
 * main - UDP Thread Per Request Server for file transfer service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: tcpTPRServer [port]\n");
    }
    /* get server socket */
    sock = passivesock(service, "tcp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++udpFileClient.c++++++++++++++++++++++++
/***********************************************************************/
/* Author: Aaron Lam                                                   */
/* Description: UDP File Client, connect to UDP file server            */
/*     get file path from user, then send request to server            */
/*     get file name return from server, create file with file name    */
/*     get file content from server and write to local file            */
/***********************************************************************/


/* udpFileClient.c - main */

#include "errexit.h"
#include "connectsock.h"

#include <unistd.h>
#include <fcntl.h>
#include <string.h>

/**
 * sendFileRequest: get filepath from user, send filepath to server using socket
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - file path, using default BUFSIZ
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
int sendFileRequest (int sock, char *buf) {
    memset(buf, 0, BUFSIZ);
    printf("File name: ");
    fgets(buf, BUFSIZ, stdin);
    buf[strlen(buf) - 1] = '\0';

    if (sendto(sock, buf, strlen(buf), 0, NULL, 0) < 0) {
        errexit("sendto() error: %s\n", strerror(errno));
    }
    return 0;
}

/**
 * createFile: get file name from server, create a file with that name
 * Arguements:
```

```
 * sock     - socket descripter, current open socket
 * buf      - storage filename recv from server, using default BUFSIZ
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
int createFile (int sock, char *buf) {
    memset(buf, 0, BUFSIZ);
    if ((recvfrom(sock, buf, BUFSIZ, 0, NULL, 0)) > 0) {
        printf("createFile: %s\n", buf);
        mode_t mode = 0666;
        int fd = open(buf, O_WRONLY | O_CREAT, mode);
        if ( fd == -1 ) {
            errexit("open() error: %s\n", strerror(errno));
        }
        return fd;
    }
    return -1;
}

/**
 * getFile: get file content from server, then write content to file in fd
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - storage filename recv from server, using default BUFSIZ
 *          - use as buf for message get from server
 * fd       - file descriptor, current open file
 * Return:
 * return nothing. If error, print error on exit
 * @author: Aaron Lam
 */
int getFile (int sock, char *buf, int fd) {
    printf("start get file ...\n");
    if (fd == -1) return -1;
    int n;
    while ( (n = recvfrom(sock, buf, BUFSIZ, 0, NULL, NULL)) > 0) {
        if (write (fd, buf, n) < 0 ) {
            errexit("write() error: %s\n", strerror(errno));
        }
    }
    if (n == -1) {
        errexit("recvfrom() error: %s\n", strerror(errno));
    }
    close(fd);
    printf("done\n");
    return 0;
}

void run_client(int sock) {
    char buf[BUFSIZ];
    int fd;
    if (sendFileRequest(sock, buf) == -1) {
        errexit("sendFileRequest() error: %s\n", strerror(errno));
    }

    if ((fd = createFile(sock, buf)) == -1){
        errexit("createFile() error: %s\n", strerror(errno));
    }

    if (getFile(sock, buf, fd) == -1) {
        errexit("getFile() error: %s\n", strerror(errno));
    }
}

/*------------------------------------------------------------------------
 * main - UDP file client for udp file service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    char *host = "localhost"; /* host to use if none supplied */
    char *service = "9000"; /* default service name */

    switch (argc) {
        case 1:
        host = "localhost";
        break;
        case 3:
        service = argv[2];
        /* FALL THROUGH */
        case 2:
        host = argv[1];
        break;
        default:
        fprintf(stderr, "usage: udpFileClient [host [port]]\n");
        return 1;
    }

    int sock = connectsock(host, service, "udp");

    run_client(sock);

    return 0;
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++udpFileServer.c++++++++++++++++++++++++
#include "udpFileServer.h"

/**
 * @author: Aaron Lam
 */

/**
 * getFilePath: get file path from client connected to current socket.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - source address, will be filled by this function
 *          - src_addr need to be saved for later use
 * socklen  - socket length, size of src_addr
 * Return:
 * the requested file path from client is save buf for later use
 * return nothing. Report error code (errno) if recvfrom() fail
 * @author: Aaron Lam
 */
void getFilePath (int sock, char *buf, struct sockaddr_in *src_addr, socklen_t
*socklen) {
    /* udp recvfrom: get data from socket */
    if (recvfrom(sock, buf, BUFSIZ, 0, (struct sockaddr *) src_addr, socklen) <
0) {
        printf("recvfrom() error: %s\n", strerror(errno));
    }
}

/**
 * sendFileName: send file name back to client, act as connection check.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - server source address, pass in as value
 * socklen  - socket length, size of src_addr
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * @author: Aaron Lam
 */
void sendFileName (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen) {
    /* get name of file from file path */
    char *filename = basename(buf);

    /* send file name back to client */
    int count = sendto(sock, filename, strlen(filename), 0, (struct sockaddr
*)&src_addr, socklen);
    if (count < 0) {
        printf("sendto() error: %s\n", strerror(errno));
    }
}

/**
 * sendFile: open the requested file, then send file back to client.
 * Arguements:
 * sock     - socket descripter, current open socket
 * buf      - buffer to storage message, using default BUFSIZ
 * src_addr - server source address, pass in as value
 * socklen  - socket length, size of src_addr
 * Return:
```

```
 * return nothing. Report error code (errno) if sendto() fail
 * @author: Aaron Lam
 */
void sendFile (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen) {
    printf("open file: %s\n", buf);
    /* open file identify by file path */
    int fd = open(buf, O_RDONLY);
    /* if open error */
    if ( fd == -1 ) {
        sendto(sock, NULL, 0, 0, (struct sockaddr *)&src_addr, socklen);
        printf("error openning file: '%s' '%s'\n", buf, strerror(errno));
    }

    printf("start sending...\n");
    /* send file content to client */
    int n;
    while ( (n = read(fd, buf, BUFSIZ)) > 0 ) {
        sendto(sock, buf, n, 0, (struct sockaddr *)&src_addr, socklen);
    }
    close(fd);
    printf("close file\n");

    /* send NULL to client to signal EOF */
    sendto(sock, NULL, 0, 0, (struct sockaddr *)&src_addr, socklen);
    printf("done sending\n");
}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++udpFileServer.h++++++++++++++++++++++++
#ifndef UDP_FILE_SERVER_H
#define UDP_FILE_SERVER_H

#include "errexit.h"
#include "passivesock.h"

#include <unistd.h>
#include <fcntl.h>
#include <libgen.h>
#include <string.h>
#include <sys/wait.h>

void getFilePath (int sock, char *buf, struct sockaddr_in *src_addr, socklen_t
*socklen);

void sendFileName (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen);

void sendFile (int sock, char *buf, struct sockaddr_in src_addr, socklen_t
socklen);

#endif
++++++++++++++++++++++++END++++++++++++++++++++++++
+++++++++++++++++++++++udpiServer.c+++++++++++++++++++++++
/* UDP_iterativeServer.c - main */

#include "errexit.h"
#include "passivesock.h"
#include "udpFileServer.h"

/**
 * run_server: do server work - get file, and send file
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 */
void run_server (int sock) {
    char buf[BUFSIZ]; /* message buffer; use default stdio BUFSIZ */

    struct sockaddr_in src_addr; /* the from address of a client */
    memset (&src_addr, 0, sizeof(src_addr));

    socklen_t socklen = sizeof(src_addr);

    while (1) {
     memset (buf, 0, BUFSIZ);

        getFilePath(sock, buf, &src_addr, &socklen);

        sendFileName(sock, buf, src_addr, socklen);

        sendFile(sock, buf, src_addr, socklen);
    }
}

/*------------------------------------------------------------------------
 * main - Iterative UDP server for file transfer service
 *------------------------------------------------------------------------
 */
int
main(int argc, char *argv[])
{
    int sock; /* server socket */
    char *service = "9000"; /* service name or port number */

    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: UDP_iterativeServer [port]\n");
    }

    sock = passivesock(service, "udp", 0);

    run_server(sock);

}
++++++++++++++++++++++++END+++++++++++++++++++++++
++++++++++++++++++++++++udpPFServer.c++++++++++++++++++++++++
/**********************************************************************/
/* Author: Aaron Lam                                                  */
/* Description: UDP file server, prefork server                       */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
/**********************************************************************/

/* udpTPRServer.c - main */

#include "errexit.h"
#include "passivesock.h"
#include "udpFileServer.h"

int CHILD_COUNT = 5;

static int main_pid;

/* fork_child: make sure only main can fork
 * Arguments: nothing
 * Return:
 * return child pid or exit if already in child process
 * @author: Aaron Lam
 */
int fork_child()
{
    if(getpid() == main_pid)
        return fork();
    else
        exit(0);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 * @author: Aaron Lam
 */
void run_server (int sock) {
    main_pid = getpid();
    int count;

    for(count = 0; count < CHILD_COUNT; ++count) {
        int pid = fork_child();
        if(pid < 0) {
            printf("fork() error: %s", strerror(errno));
        }
        else if (pid == 0) {
            /* message buffer; use default stdio BUFSIZ */
            char buf[BUFSIZ];

            /* the from address of a client */
            struct sockaddr_in src_addr;
            /* socket length */
            socklen_t socklen = sizeof(src_addr);

            int run_flag = 1;
            while (run_flag) {
                // run_flag = 0; /* for mem leak check */

                /* zero out variable before use */
                memset(buf, 0, BUFSIZ);
                memset(&src_addr, 0, sizeof(src_addr));

                /* do server work */
                getFilePath(sock, buf, &src_addr, &socklen);
                sendFileName(sock, buf, src_addr, socklen);
                sendFile(sock, buf, src_addr, socklen);
            }
            exit(0);
        }
    }

    wait(NULL);
}

/*---------------------------------------------------------------------------
 * main - UDP Prefork Server for file transfer service
 *---------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        CHILD_COUNT = atoi(argv[1]);
        break;
        case 3:
        service = argv[1];
        CHILD_COUNT = atoi(argv[2]);
        default:
        errexit("usage: udpPFServer [port] [child_num]\n");
    }

    /* get server socket */
    sock = passivesock(service, "udp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++udpPPRServer.c++++++++++++++++++++++++
/**********************************************************************/
/* Author: Aaron Lam                                                  */
/* Description: UDP file server, process per request                  */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
/**********************************************************************/

/* udpTPRServer.c - main */

#include <pthread.h>
#include <sys/wait.h>

#include "errexit.h"
#include "passivesock.h"
#include "udpFileServer.h"

static int main_pid;

/* fork_child: make sure only main can fork
 * Arguments: nothing
 * Return:
 * return child pid or exit if already in child process
 * @author: Aaron Lam
 */
int fork_child()
{
    if(getpid() == main_pid)
        return fork();
    else
        exit(0);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 */
void run_server (int sock) {
    main_pid = getpid();


    /* message buffer; use default stdio BUFSIZ */
    char buf[BUFSIZ];

    /* the from address of a client */
    struct sockaddr_in src_addr;
    /* zero out the src_addr so it can be filled later */
    memset (&src_addr, 0, sizeof(src_addr));

    socklen_t socklen = sizeof(src_addr);

    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for mem leak check */

        /* use getFilePath() as blocked read from socket */
        getFilePath(sock, buf, &src_addr, &socklen);

        int pid = fork_child();
        if(pid < 0) {
            printf("fork() error: %s", strerror(errno));
        }
        else if (pid == 0) {
            sendFileName(sock, buf, src_addr, socklen);
            sendFile(sock, buf, src_addr, socklen);
            exit(0);
        }
    }
    wait(NULL);
}

/*------------------------------------------------------------------------
 * main - UDP Thread Per Request Server for file transfer service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: udpPPTServer [port]\n");
    }

    /* get server socket */
    sock = passivesock(service, "udp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
++++++++++++++++++++++++udpPTServer.c++++++++++++++++++++++++
/**********************************************************************/
/* Author: Aaron Lam                                                  */
/* Description: UDP file server, pre-threaded (thread pool) server   */
/*  get file name from client, use a thread pool to handle requests  */
/*  the server process is keep alive in a forever while loop         */
/**********************************************************************/

/* udpPTServer.c - main */

#include <pthread.h>

#include "errexit.h"
#include "passivesock.h"
#include "udpFileServer.h"

int THREAD_COUNT = 5;

pthread_mutex_t mutex;

/**
 * handle_request: handle client request, call by pthread_create
 * Arguements:
 * input: socket that created in main thread, need to cast to int before use
 * Return:
 * return nothing.
 * get file path from client, then send file name and file content back to client
 * @author: Aaron Lam
 */
void *handle_request (void *input) {
    /* server socket descriptor */
    long *sock_pt = (long*) input;
    int sock = *sock_pt;

    /* message buffer; use default stdio BUFSIZ */
    char buf[BUFSIZ];

    /* the from address of a client */
    struct sockaddr_in src_addr;

    socklen_t socklen = sizeof(src_addr);

    int run_flag = 1;
    while (run_flag) {
        // run_flag = 0; /* for mem leak check */

        /* zero out variable before use */
        memset(buf, 0, BUFSIZ);
        memset(&src_addr, 0, socklen);

        pthread_mutex_lock(&mutex);
        getFilePath(sock, buf, &src_addr, &socklen);
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        sendFileName(sock, buf, src_addr, socklen);
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        sendFile(sock, buf, src_addr, socklen);
        pthread_mutex_unlock(&mutex);

    }
    pthread_exit(NULL);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error code (errno) if sendto() fail
 * server will not terminate if error happen, only print error
 * @author: Aaron Lam
 */
void run_server (int sock) {

    pthread_t thread_ids[THREAD_COUNT];

    pthread_mutex_init(&mutex, NULL);
    pthread_attr_t attr;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    int i;
    for(i = 0; i < THREAD_COUNT; ++i) {
        pthread_create(&thread_ids[i], &attr, handle_request, (void *) &sock);
    }

    pthread_attr_destroy(&attr);
    for(i = 0; i < THREAD_COUNT; ++i) {
        pthread_join(thread_ids[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    pthread_exit(NULL);
}

/*------------------------------------------------------------------------
 * main - UDP Pre-Thread Server for file transfer service
 *------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        /* defaut case: port 9000, thread count 5 */
        case 1:
        break;
        case 2:
        THREAD_COUNT = atoi(argv[1]);
        break;
        case 3:
        service = argv[1];
        THREAD_COUNT = atoi(argv[2]);
        break;
        default:
        errexit("usage: udpPTServer [port] [thread_count]\n");
    }

    /* get server socket */
    sock = passivesock(service, "udp", 0);

    /* run server */
    run_server(sock);

}
+++++++++++++++++++++++END+++++++++++++++++++++++
++++++++++++++++++++++++udpTPRServer.c+++++++++++++++++++++++
/*****************************************************************/
/* Author: Aaron Lam                                               */
/* Description: UDP file server, thread per request                */
/*  get file name from client, then creat a thread to handle request */
/*  the server process is keep alive in a forever while loop         */
```

```
/**********************************************************************/

/* udpTPRServer.c - main */

#include <pthread.h>

#include "errexit.h"
#include "passivesock.h"
#include "udpFileServer.h"

typedef struct {
    int sock;
    char buf[BUFSIZ];
    struct sockaddr_in src_addr;
    int socklen;
} ThreadData;

/**
 * handle_request: handle client request, call by pthread_create
 * Arguements:
 * input: storage ThreadData, need to be covert to ThreadData before use
 * Return:
 * return nothing. send file name and file content back to client
 * @author: Aaron Lam
 */
void *handle_request (void *input) {
    ThreadData *threadData = (ThreadData*) input;

    int sock = threadData->sock;
    char* buf = threadData->buf;
    struct sockaddr_in src_addr = threadData->src_addr;
    int socklen = threadData->socklen;

    sendFileName(sock, buf, src_addr, socklen);

    sendFile(sock, buf, src_addr, socklen);

    pthread_exit(NULL);
}

/**
 * run_server: do server work - get file, and send file
 * use getFilePath() as blocking function before creating thread
 * Arguements:
 * sock     - socket descripter, current open socket
 * Return:
 * return nothing. Report error message (errno) if error
 * @author: Aaron Lam
 */
void run_server (int sock) {

    /* thread id */
    pthread_t thread;
    /* thread attribule */
    pthread_attr_t attr;

    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    /* set detach state so main don't have to call pthread_join */
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    /* while loop to keep server alive
     * variables needed for a thread are created in while
     * no share variables between threads except socket
     */
    int run_flag = 1;
    while (run_flag) {

        // run_flag = 0; /* for mem leak check */

        /* message buffer; use default stdio BUFSIZ */
        char buf[BUFSIZ];

        /* the from address of a client */
        struct sockaddr_in src_addr;
        /* zero out the src_addr so it can be filled later */
        memset (&src_addr, 0, sizeof(src_addr));

        socklen_t socklen = sizeof(src_addr);

        /* use getFilePath() as blocked read from socket */
        getFilePath(sock, buf, &src_addr, &socklen);

        /* struct data to be pass to thread*/
        ThreadData threadData;
        /* fill in threadDate with data get back for getFilePath() */
        threadData.sock = sock;
        strcpy(threadData.buf, buf);
        threadData.src_addr = src_addr;
        threadData.socklen =socklen;

        /* create thread to handle request */
        if (pthread_create(&thread, &attr, handle_request, (void*) &threadData) <
0) {
            printf("pthread_create(): %s\n", strerror(errno));
        }
    }

    /* destroy attribute to void memory leak */
    pthread_attr_destroy(&attr);

    /* main thread exit */
    pthread_exit(NULL);
}
```

```
/*--------------------------------------------------------------------------
 * main - UDP Thread Per Request Server for file transfer service
 *--------------------------------------------------------------------------
 */
int main(int argc, char *argv[]) {
    /* server socket */
    int sock;
    /* default port number */
    char *service = "9000";

    /* get command line input */
    switch (argc) {
        case 1:
        break;
        case 2:
        service = argv[1];
        break;
        default:
        errexit("usage: udpTPRServer [port]\n");
    }

    /* get server socket */
    sock = passivesock(service, "udp", 0);

    /* run server */
    run_server(sock);

}
++++++++++++++++++++++++END++++++++++++++++++++++++
```