Ονοματεπώνυμο: Παντελεήμων Μαλέκας
A.M: 1115201600268

A) Dropbox_server

Compilation command 1: make (with Makefile, also compiles dropbox_client)
Compilation command 2: gcc -o dropbox_server dropbox_server.c Client_List.c (without Makefile)

Run command: ./dropbox_server -p portNum

In order for the program to work the arguments must be given in that order.

After getting the parameter the program gets server's IP and then creates the socket where he will receive messages from clients. A server may receive three types of messages:

1.) LOG_ON IP PORT: Sent by a client who just logged on to the system. The server will add him to a simple linked list and will also send the USER_ON IP PORT message to the other clients who may be already logged on.
2.) GET_CLIENTS: Sent by a client to find out which other clients are already in the system. In my implementation I also send the IP and PORT of the one who sent the request so the server will know to send back the IPs and PORTs of the other clients only. The message that is sent back is CLIENT_LIST N IP1 PORT1 … IPN PORTN.
3.) LOG_OFF IP PORT: Sent by a client who exits the system. The server will remove him from his list and will send the USER_OFF message if there are any other clients logged on in the system. In case the IP and PORT are not found in the list the program will print ERROR_IP_PORT_NOT_FOUND_IN_LIST

Whenever an IP and PORT are sent throughout both applications, functions inet_pton and htons are used to turn IPs and PORTs respectively, to binary formats.

Also select function is used so the server will know which client it will serve first.

Finally the server exits when SIGINT signal is received, releasing all possible allocated memory.

B) Dropbox_client

Compilation command 1: make (with Makefile, also compiles dropbox_server)
Compilation command 2: gcc -o dropbox_client dropbox_client.c Client_List.c Buffer.c Functions.c -lpthread (without Makefile)

Run command: ./dropbox_client -d dirName -p portNum -w workerThreads -b bufferSize -sp serverPort -sip serverIP
In order for the program to work the arguments must be given in that order.
Also, please type only the names of the desired folders. For example: “input1”. Don’t type “./input1”, or the path to it, since all the necessary work regarding paths is handled by the program.

After getting the parameters, the program creates the necessary directory and file paths that will be used throughout the program. Then the circular buffer is created, the client’s IP and port are turned into binary format, and all the work for the socket fds is done.

The circular buffer is a struct which consists of an array of struct buffer_item items, two pointers which point at the start and end of the buffer and a counter. Struct buffer_item has four fields: IP, PORT, PATHNAME, VERSION. Default values are NULL, 0, NULL,-1 respectively.

After all the initial work, the client sends to the server the LOG_ON message to make his presence known and the GET_CLIENTS message to receive the IPs and PORTs from clients who may already be in the system. After this, the worker threads are created and the IPs and PORTs are placed in the buffer.

Then the main thread enters a server-like while 1 loop to receive messages from the server or other clients. The main thread may receive four types of messages:

1.) GET_FILE_LIST: When a client receives this message he sends all the pathnames from the files found in his input directory. The message that will be sent back is FILE_LIST N PATHNAME1 VERSION1 … PATHNAMEN VERSIONN. In my implementation I also send the pathname of the input directory (separately) because the worker thread needs it.
2.) GET_FILE PATHNAME VERSION: When a client receives this message, first he checks if the file exists. If it doesn’t, the message FILE_NOT_FOUND will be sent. Otherwise, we get its version and compare it with the one sent by the worker thread to see what message will be sent. For the file version I decided to hash the content of the file with its size in bytes. If the versions are the same, then the FILE_UP_TO_DATE is sent. Otherwise the file is sent with the message: FILE_SIZE VERSION N BYTE0BYTE1...BYTEN.
3.) USER_OFF IP PORT: Sent by the server whenever a client exits the system. The main thread simply removes him from the client list.
4.) USER_ON IP PORT: Sent by the server whenever a client enters the system. The main thread places him in the client list and the circular buffer.

The select function is used so the client will know which message to serve first.

Each worker thread works as follows: If the buffer has at least one item, the thread will remove one item. There are two types of items the thread needs to take care of:

If the item is IP and PORT (which means PATHNAME and VERSION fields have their default values), the thread will send a GET_FILE_LIST message to get all the pathnames. If the path leads to a directory the thread creates the respective path in the mirror folder and doesn't place it in the buffer because it is not needed. If the path leads to a regular file, the thread places it in the buffer. The mirror folder where the received files are placed are stored in a Clients folder which is created in the input directory. This folder has a clientIP_clientPORT folder for each sender, and this is where the files are placed. After placing all the files in the buffer, the thread enters a while 1 loop to get an item if the buffer isn't empty.

If the item is IP, PORT, PATHNAME and VERSION the thread will check if the file exists in the respective mirror folder. If the file doesn't exist the version that will be sent is set to -1 and the message GET_FILE PATHNAME VERSION is sent in order to get the file. If the file does exist we get its version and we sent the message GET_FILE PATHNAME VERSION to see whether we need to get an updated version or not. If we get FILE_SIZE VERSION N BYTES, this means the file changed so we copy it. If we get FILE_UP_TO_DATE, we simply a print an appropriate message. After all the necessary work is done the thread enters the while 1 loop to get an item if the buffer isn't empty.

A worker thread exits when the main thread exits the system by receiving from it a SIGUSR1 signal.

The main thread exits when SIGINT signal is received, sending the LOG_OFF message to the server, releasing possible allocated memory and finishing off the worker threads.