

Gebze Technical University

Computer Engineering Faculty

Final Project

System Programming



Student: Burak Ersoy

No: 1901042703

BibakBOXServer

The provided multi-threaded file server code demonstrates a basic implementation of a server that can handle multiple client connections concurrently. It allows clients to upload, delete, and update files in their respective directories, while also providing synchronization of directory contents upon connection. The server employs thread pooling for efficient resource utilization and includes error handling and signal handling mechanisms for robustness.

Function Explanation

```
// Function to write log entry to the client's logfile
void writeLog(const char* clientDir, const char* message)
{
    char logFile[BUFFER_SIZE];
    sprintf(logFile, "%s/logfile.txt", clientDir);

    FILE* log = fopen(logFile, "a");
    if (log == NULL)
    {
        perror("Error opening logfile");
        return;
    }

    time_t now;
    time(&now);
    char* timestamp = ctime(&now);
    timestamp[strlen(timestamp) - 1] = '\0'; // Remove newline character from ctime result

    pthread_mutex_lock(&logMutex);
    fprintf(log, "[%s] %s\n", timestamp, message);
    pthread_mutex_unlock(&logMutex);

    fclose(log);
}

// Function to synchronize directory contents with the client
void synchronizeDirectory(int clientSocket, const char* clientDir)
{
    char buffer[BUFFER_SIZE];

    DIR* dir = opendir(clientDir);
    if (dir == NULL)
    {
        perror("Error opening directory");
        return;
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL)
    {
        if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0)
        {
            strcpy(buffer, entry->d_name);
            send(clientSocket, buffer, strlen(buffer), 0);
            memset(buffer, 0, sizeof(buffer));
        }
    }

    closedir(dir);
}
```

1- writeLog(const char* clientDir, const char* message):

This function is responsible for writing log entries to the client's logfile. It receives the client's directory and the message to be logged. It opens the client's logfile in "a" (append) mode, retrieves the current timestamp, and writes the timestamp and message to the logfile. It uses a mutex (logMutex) to ensure thread safety when writing to the logfile.

2-synchronizeDirectory(int clientSocket, const char* clientDir):

This function synchronizes the directory contents with the client. It takes the client socket and the client's directory as input. It opens the client's directory using opendir and iterates through the directory entries using readdir. For each file entry, excluding the "." and ".." directories, it sends the file name to the client using the send function. It uses a buffer (buffer) to hold the file name temporarily.

```
// Function to handle a client connection
void* handleClient(void* arg)
{
    int clientSocket = *((int*)arg);
    char buffer[BUFFER_SIZE];

    // Receive client directory name
    ssize_t numBytes = recv(clientSocket, buffer, sizeof(buffer), 0);
    if (numBytes <= 0)
    {
        close(clientSocket);
        return NULL;
    }

    buffer[numBytes] = '\0';
    char clientDir[1024];
    snprintf(clientDir, sizeof(clientDir), "%s/%s", directory, buffer);

    // Synchronize directory with client
    synchronizeDirectory(clientSocket, clientDir);

    printf("Client connected: %s\n", buffer);

    while (1)
    {
        // Receive client requests
        numBytes = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (numBytes <= 0)
        {
            break;
        }

        // Handle file operations
        buffer[numBytes] = '\0';
        char* operation = strtok(buffer, ":");
        char* filename = strtok(NULL, ":");

        if (strcmp(operation, "upload") == 0)
        {
            FILE* file = fopen(filename, "wb");
            if (file == NULL)
            {
                perror("Error opening file");
                continue;
            }

            while ((numBytes = recv(clientSocket, buffer, sizeof(buffer), 0)) > 0)
            {
                fwrite(buffer, 1, numBytes, file);
            }

            fclose(file);

            printf("File uploaded: %s\n", filename);
            writelog(clientDir, filename);
        }
        else if (strcmp(operation, "delete") == 0)
        {
            char filePath[1024];
            sprintf(filePath, "%s/%s", clientDir, filename);
            if (remove(filePath) == 0)
            {
                printf("File deleted: %s\n", filename);
                writelog(clientDir, filename);
            }
            else
            {
                perror("Error deleting file");
            }
        }
        else if (strcmp(operation, "update") == 0)
        {
            // Currently, updating a file means deleting the old file and uploading the new one
            char filePath[1024];
            sprintf(filePath, "%s/%s", clientDir, filename);
            if (remove(filePath) == 0)
            {
                FILE* file = fopen(filename, "wb");
                if (file == NULL)
                {
                    perror("Error opening file");
                    continue;
                }

                while ((numBytes = recv(clientSocket, buffer, sizeof(buffer), 0)) > 0)
                {
                    fwrite(buffer, 1, numBytes, file);
                }

                fclose(file);
                printf("File updated: %s\n", filename);
                writelog(clientDir, filename);
            }
            else
            {
                perror("Error updating file");
            }
        }
    }

    printf("Client disconnected: %s\n", buffer);

    pthread_mutex_unlock(&client_mutex);
    close(clientSocket);
    free(arg);
}
```

```
// Signal handler for SIGINT
void handleSIGINT(int signum)
{
    printf("\nTerminating server...\n");

    // Close the server socket
    close(serverSocket);

    // Wait for all client threads to finish
    pthread_mutex_lock(&logMutex);
    printf("Waiting for all client threads to finish...\n");
    pthread_mutex_unlock(&logMutex);

    pthread_exit(NULL);
    exit(EXIT_SUCCESS);
}
```

3- handleClient(void* arg):

This function is the main function executed by each client thread. It takes a void* argument as input, which is cast to an integer representing the client socket. The function starts by receiving the client directory name from the client using recv. It constructs the full client directory path by appending the received directory name to the server's root directory. Then, it calls the

synchronizeDirectory function to synchronize the directory contents with the client. After the directory synchronization, the function enters a loop to handle client requests. It receives client requests using recv and tokenizes the received message to extract the operation and filename. Based on the operation type, it performs the requested file operation:

For "upload", it opens the specified file in write-binary mode ("wb"), receives the file data from the client using recv, and writes the received data to the file using fwrite. It also prints a message and writes a log entry for the uploaded file.

For "delete", it constructs the full file path by appending the filename to the client's directory and deletes the file using remove. It prints a message and writes a log entry for the deleted file.

For "update", it deletes the existing file (if it exists) using remove, creates a new file with the same name in write-binary mode, receives the updated file data from the client using recv, and writes the received data to the new file. It prints a message and writes a log entry for the updated file.

4- handleSIGINT(int signum): This function is the signal handler for the SIGINT signal (Ctrl+C). It is called when the server receives the SIGINT signal, indicating a termination request. The function prints a termination message, closes the server socket, and waits for all client threads to finish by acquiring the log mutex (logMutex). Finally, it exits the program using pthread_exit and exit with a successful status.

BibakBOXClient

The program is to be a client program for a directory synchronization application. It allows the user to upload, delete, and update files in a specified directory, while keeping the local directory synchronized with a remote server.

Function Explanation

```
// Function to synchronize directory contents with the server
void synchronizeDirectory(int serverSocket)
{
    char buffer[BUFFER_SIZE];

    DIR* dir = opendir(clientDir);
    if (dir == NULL) {
        perror("Error opening directory");
        return;
    }

    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL)
    {
        if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0)
        {
            strcpy(buffer, entry->d_name);
            send(serverSocket, buffer, strlen(buffer), 0);
            memset(buffer, 0, sizeof(buffer));
        }
    }

    closedir(dir);
}
```

```
// Function to send file to the server
void sendFile(int serverSocket, const char* filePath)
{
    char buffer[BUFFER_SIZE];

    FILE* file = fopen(filePath, "rb");
    if (file == NULL)
    {
        perror("Error opening file");
        return;
    }

    send(serverSocket, "upload:", strlen("upload:"), 0);
    send(serverSocket, filePath, strlen(filePath), 0);

    while (1)
    {
        size_t bytesRead = fread(buffer, 1, sizeof(buffer), file);
        if (bytesRead > 0)
        {
            send(serverSocket, buffer, bytesRead, 0);
        }

        if (bytesRead < sizeof(buffer))
        {
            if (feof(file)) {
                break;
            } else if (ferror(file)) {
                perror("Error reading from file");
                break;
            }
        }
    }

    fclose(file);
}
```

1- synchronizeDirectory(int serverSocket):

This function is responsible for synchronizing the client's directory with the server. It opens the client directory, reads its contents, and sends the filenames to the server using the send function.

It's important to note that this function does not handle the actual transfer of file contents. It only sends the file names to the server. The transfer of file contents is handled separately in the sendFile function.

2- void sendFile(int serverSocket, const char* filePath)

The sendFile function is responsible for sending a file to the server. It takes two parameters: serverSocket, which represents the socket descriptor for the server connection, and filePath, which is the path to the file that needs to be sent.

The purpose of this function is to read the contents of a file in chunks and send them to the server over the network connection. By breaking the file into smaller chunks, it allows for efficient transmission and minimizes memory usage.

It's important to note that this function assumes the file has already been opened for reading and that the upload command and file path have been sent to the server before calling sendFile.


```
// Function to handle directory synchronization
void* synchronize(void* arg)
{
    int serverSocket = *((int*)arg);
    char buffer[BUFFER_SIZE];

    // Send client directory name to the server
    send(serverSocket, clientDir, strlen(clientDir), 0);

    // Synchronize directory with server
    synchronizeDirectory(serverSocket);

    while (1)
    {
        // Monitor local directory for changes
        DIR* dir = opendir(clientDir);
        if (dir == NULL)
        {
            perror("Error opening directory");
            break;
        }

        struct dirent* entry;
        while ((entry = readdir(dir)) != NULL)
        {
            if (strcmp(entry->d_name, ".") != 0 && strcmp(entry->d_name, "..") != 0) {
                char filePath[BUFFER_SIZE];
                sprintf(filePath, "%s/%s", clientDir, entry->d_name);

                // Check if the file exists on the server
                send(serverSocket, "check:", strlen("check:"), 0);
                send(serverSocket, entry->d_name, strlen(entry->d_name), 0);

                ssize_t numBytes = recv(serverSocket, buffer, sizeof(buffer), 0);
                if (numBytes <= 0)
                {
                    perror("Error receiving data from server");
                    break;
                }

                buffer[numBytes] = '\0';

                if (strcmp(buffer, "exists") != 0)
                {
                    // File doesn't exist on the server, so send it
                    sendFile(serverSocket, filePath);
                }
                else
                {
                    // File exists on the server, check if it's up to date
                    send(serverSocket, "timestamp:", strlen("timestamp:"), 0);
                    send(serverSocket, entry->d_name, strlen(entry->d_name), 0);

                    numBytes = recv(serverSocket, buffer, sizeof(buffer), 0);
                    if (numBytes <= 0)
                    {
                        perror("Error receiving data from server");
                        break;
                    }

                    buffer[numBytes] = '\0';

                    long clientTimestamp = strtol(buffer, NULL, 10);
                    struct stat fileStat;
                    stat(filePath, &fileStat);
                    long serverTimestamp = fileStat.st_mtime;

                    if (clientTimestamp > serverTimestamp)
                    {
                        // Client file is newer, so send it
                        sendFile(serverSocket, filePath);
                    }
                }
            }
        }

        closedir(dir);

        sleep(5); // Check for changes every 5 seconds
    }

    close(serverSocket);
    free(arg);
    return NULL;
}
```

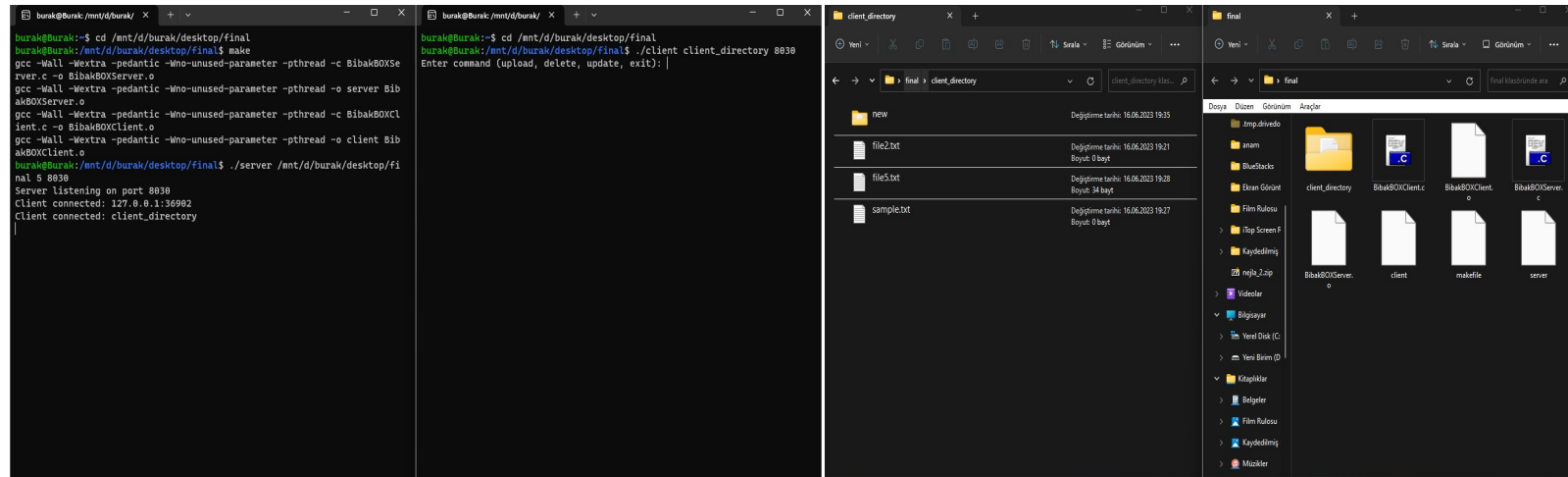
3- synchronize(void* arg):

This function serves as the thread function for directory synchronization. It receives a socket descriptor as an argument and performs the following tasks in an infinite loop:

- Sends the client directory name to the server.
- Calls the synchronizeDirectory function to sync the directory contents with the server.
- Monitors the local directory for changes by repeatedly opening it, reading the file names, and comparing them with the server's records.
- If a file doesn't exist on the server, it calls sendFile to upload the file.
- If a file exists on the server, it compares the timestamps to determine if an update is required.
- The loop sleeps for 5 seconds between checks.

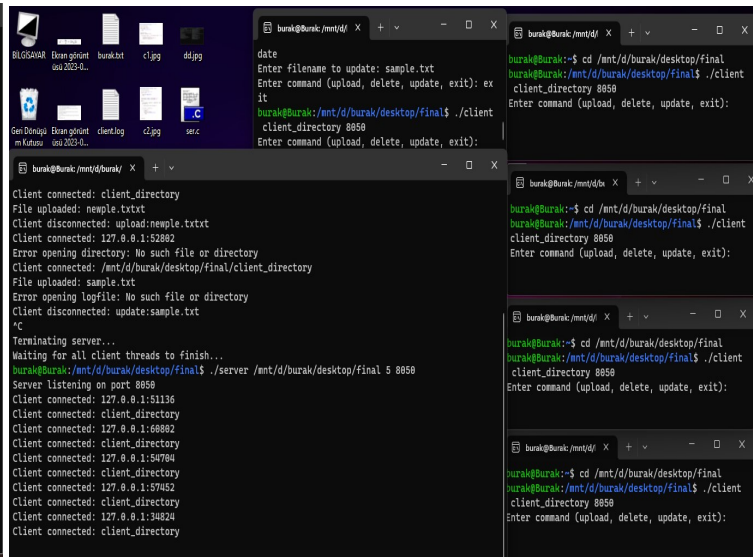
The purpose of this function is to synchronize the client's directory with the server by comparing the files in the client's directory with the files on the server. It sends new or modified files to the server and updates the server with the latest versions of the files.

TEST CASES



```
burak@Burak:~/mnt/d/burak/ X + v - □ X
burak@Burak:~$ cd /mnt/d/burak/desktop/final
burak@Burak:~/mnt/d/burak/desktop/final$ make
gcc -Wall -Wextra -pedantic -Wno-unused-parameter -pthread -c BibakBOXSe
rver.c -o BibakBOXServer.o
gcc -Wall -Wextra -pedantic -Wno-unused-parameter -pthread -o server Bib
akBOXServer.o
gcc -Wall -Wextra -pedantic -Wno-unused-parameter -pthread -c BibakBOXCl
ient.c -o BibakBOXClient.o
gcc -Wall -Wextra -pedantic -Wno-unused-parameter -pthread -o client Bib
akBOXClient.o
burak@Burak:~/mnt/d/burak/desktop/final$ ./server /mnt/d/burak/desktop/fi
nal 5 8040
Server listening on port 8040
Client connected: 127.0.0.1:38918
Client connected: client_directory
File uploaded: newple.txtxt
Client disconnected: upload:newple.txtxt
Client connected: 127.0.0.1:52802
Error opening directory: No such file or directory
Client connected: /mnt/d/burak/desktop/final/client_directory
File uploaded: sample.txt
Error opening logfile: No such file or directory
Client disconnected: update:sample.txt

burak@Burak:~$ cd /mnt/d/burak/desktop/final
burak@Burak:~/mnt/d/burak/desktop/final$ ./client client_directory 8040
Enter command (upload, delete, update, exit): upload
Enter filename to upload: testcase.txt
Error opening file: No such file or directory
Enter command (upload, delete, update, exit): upload
Enter filename to upload: sample.txt
Error opening file: No such file or directory
Enter command (upload, delete, update, exit): exit
burak@Burak:~/mnt/d/burak/desktop/final$ ./client /mnt/d/burak/desktop/fi
nal/client_directory 8040
Enter command (upload, delete, update, exit): upload
Enter filename to upload: sample.txt
Error opening file: No such file or directory
Enter command (upload, delete, update, exit): delete
Enter filename to delete: sample.txt
Enter command (upload, delete, update, exit): update
Invalid command
Enter command (upload, delete, update, exit): update
Enter filename to update: sample.txt
Enter command (upload, delete, update, exit): exit
burak@Burak:~/mnt/d/burak/desktop/final$
```



```
date
Enter filename to update: sample.txt
Enter command (upload, delete, update, exit): ex
it
burak@Burak:~/mnt/d/burak/desktop/final$ ./client
client_directory 8050
Enter command (upload, delete, update, exit):

burak@Burak:~/mnt/d/burak/desktop/final$ ./server /mnt/d/burak/desktop/final 5 8050
Server listening on port 8050
Client connected: 127.0.0.1:11116
Client connected: client_directory
Client connected: 127.0.0.1:60002
Client connected: client_directory
Client connected: 127.0.0.1:50704
Client connected: client_directory
Client connected: 127.0.0.1:57052
Client connected: client_directory
Client connected: 127.0.0.1:34824
Client connected: client_directory

burak@Burak:~/mnt/d/burak/desktop/final$ ./client
client_directory 8050
Enter command (upload, delete, update, exit):

burak@Burak:~/mnt/d/burak/desktop/final$ ./client
client_directory 8050
Enter command (upload, delete, update, exit):
```

