**Assignment - 08**

Question No – 01

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <pwd.h>

typedef struct {

char username[256];

off\_t filesize;

} FileInfo;

FileInfo \*fileInfos = NULL;

size\_t fileCount = 0;

// Function to add file information

void addFileInfo(const char \*username, off\_t filesize) {

fileInfos = realloc(fileInfos, (fileCount + 1) \* sizeof(FileInfo));

if (fileInfos == NULL) {

perror("Failed to allocate memory");

exit(EXIT\_FAILURE);

}

strcpy(fileInfos[fileCount].username, username);

fileInfos[fileCount].filesize = filesize;

fileCount++;

}

// Map function: Collect file sizes and owners

void mapFiles() {

DIR \*dir;

struct dirent \*entry;

struct stat fileStat;

struct passwd \*pw;

dir = opendir(".");

if (dir == NULL) {

perror("Failed to open directory");

exit(EXIT\_FAILURE);

}

while ((entry = readdir(dir)) != NULL) {

if (entry->d\_type == DT\_REG) { // Only regular files

if (stat(entry->d\_name, &fileStat) == 0) {

pw = getpwuid(fileStat.st\_uid);

if (pw != NULL) {

addFileInfo(pw->pw\_name, fileStat.st\_size);

}

}

}

}

closedir(dir);

}

// Reduce function: Find maximum size and corresponding users

void reduceFiles() {

off\_t maxSize = 0;

char \*users = NULL;

for (size\_t i = 0; i < fileCount; i++) {

if (fileInfos[i].filesize > maxSize) {

maxSize = fileInfos[i].filesize;

free(users);

users = strdup(fileInfos[i].username);

} else if (fileInfos[i].filesize == maxSize) {

// Add user to the list if not already there

if (strstr(users, fileInfos[i].username) == NULL) {

char \*newUsers = malloc(strlen(users) + strlen(fileInfos[i].username) + 2);

if (newUsers == NULL) {

perror("Failed to allocate memory");

exit(EXIT\_FAILURE);

}

sprintf(newUsers, "%s,%s", users, fileInfos[i].username);

free(users);

users = newUsers;

}

}

}

if (maxSize > 0) {

printf("Maximum file size: %ld bytes\n", maxSize);

printf("Users owning the largest file: %s\n", users);

} else {

printf("No files found.\n");

}

free(users);

}

int main() {

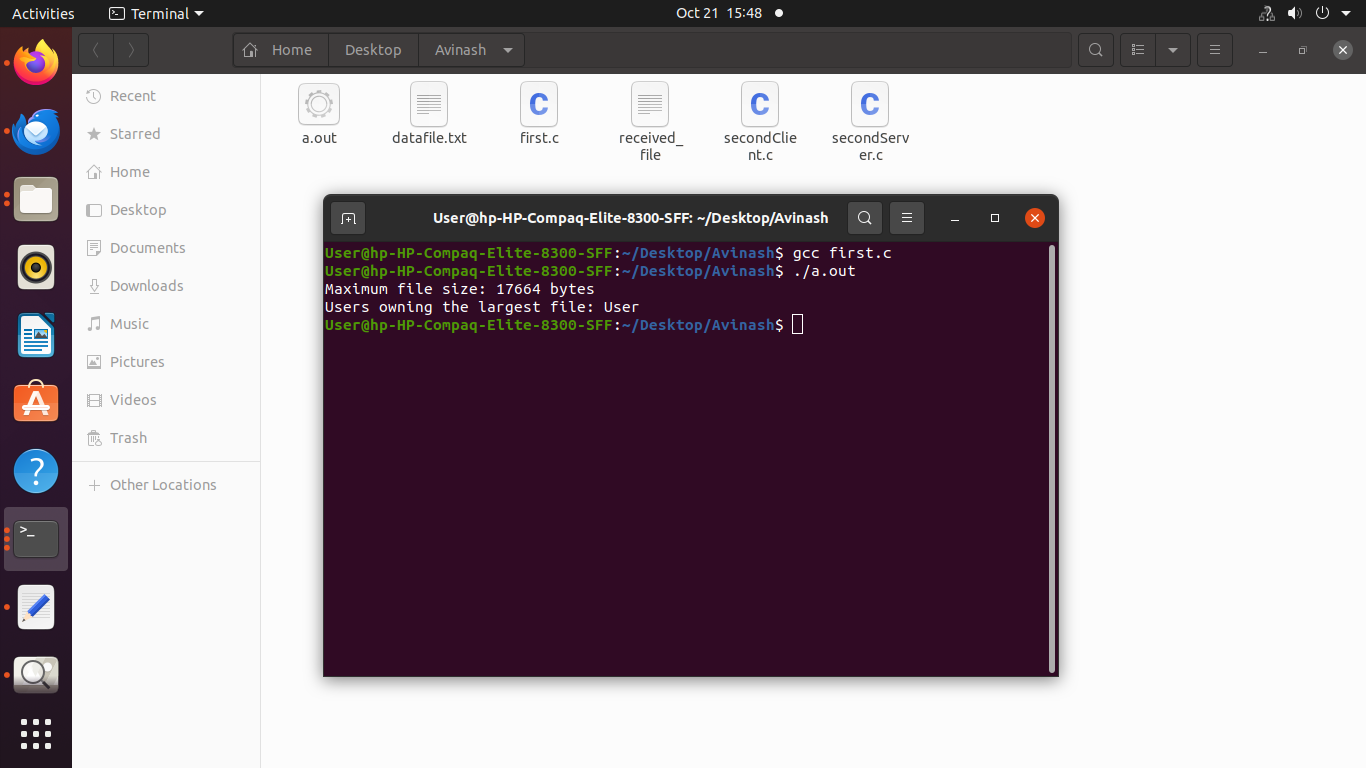
mapFiles();

reduceFiles();

free(fileInfos);

return 0;

}



Question No – 02

**[Client]**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void send\_file(const char \*filename, int socket) {

char buffer[BUFFER\_SIZE];

FILE \*fp;

size\_t n;

// Open the file for reading

fp = fopen(filename, "rb");

if (fp == NULL) {

perror("File open error");

return;

}

// Send the file in chunks

while ((n = fread(buffer, sizeof(char), BUFFER\_SIZE, fp)) > 0) {

if (send(socket, buffer, n, 0) < 0) {

perror("Send error");

break;

}

}

printf("File sent successfully.\n");

fclose(fp);

}

int main(int argc, char \*argv[]) {

int sock;

struct sockaddr\_in server\_address;

if (argc < 2) {

fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

return 1;

}

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation error");

return -1;

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(PORT);

server\_address.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Server IP

// Connect to the server

if (connect(sock, (struct sockaddr \*)&server\_address, sizeof(server\_address)) < 0) {

perror("Connection error");

return -1;

}

send\_file(argv[1], sock);

close(sock);

return 0;

}

**[Server]**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void receive\_file(int new\_socket) {

char buffer[BUFFER\_SIZE];

FILE \*fp;

int n;

// Open the file for writing

fp = fopen("received\_file", "wb");

if (fp == NULL) {

perror("File open error");

return;

}

// Receive the file in chunks

while ((n = recv(new\_socket, buffer, BUFFER\_SIZE, 0)) > 0) {

fwrite(buffer, sizeof(char), n, fp);

}

if (n < 0) {

perror("Receive error");

}

printf("File received successfully.\n");

fclose(fp);

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

// Create socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation error");

exit(EXIT\_FAILURE);

}

// Bind the socket to the port

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt))) {

perror("Set socket options error");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind error");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen error");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

// Accept an incoming connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept error");

exit(EXIT\_FAILURE);

}

receive\_file(new\_socket);

close(new\_socket);

close(server\_fd);

return 0;

}

