File Manipulation using System Calls in C++ on Linux

Objective:

Create a C++ program that performs file manipulation using Linux system calls. The program should be able to:

Create a new file.

Write a specified string to the file.

Read the contents of the file and display them on the console.

Append additional text to the file.

Delete the file.

Requirements:

Use system calls like open, read, write, close, and unlink.

Handle errors appropriately by checking the return values of system calls and using perror to print error messages.

Ensure the program is modular with separate functions for each file operation (create, write, read, append, delete).

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/types.h>

int createFile(const char \*fileName) {

int fd = open(fileName, O\_CREAT | O\_WRONLY, 0644);

if (fd == -1) {

perror("Error creating file");

return -1;

}

std::cout << "File created successfully: " << fileName << std::endl;

close(fd);

return 0;

}

int writeFile(const char \*fileName, const char \*data) {

int fd = open(fileName, O\_WRONLY | O\_TRUNC);

if (fd == -1) {

perror("Error opening file for writing");

return -1;

}

ssize\_t bytesWritten = write(fd, data, strlen(data));

if (bytesWritten == -1) {

perror("Error writing to file");

close(fd);

return -1;

}

std::cout << "Data written to file: " << data << std::endl;

close(fd);

return 0;

}

int readFile(const char \*fileName) {

int fd = open(fileName, O\_RDONLY);

if (fd == -1) {

perror("Error opening file for reading");

return -1;

}

char buffer[1024];

ssize\_t bytesRead;

while ((bytesRead = read(fd, buffer, sizeof(buffer) - 1)) > 0) {

buffer[bytesRead] = '\0';

std::cout << "File content:\n" << buffer;

}

if (bytesRead == -1) {

perror("Error reading file");

}

close(fd);

return 0;

}

int appendFile(const char \*fileName, const char \*data) {

int fd = open(fileName, O\_WRONLY | O\_APPEND);

if (fd == -1) {

perror("Error opening file for appending");

return -1;

}

ssize\_t bytesWritten = write(fd, data, strlen(data));

if (bytesWritten == -1) {

perror("Error appending to file");

close(fd);

return -1;

}

std::cout << "Data appended to file: " << data << std::endl;

close(fd);

return 0;

}

int deleteFile(const char \*fileName) {

int status = unlink(fileName);

if (status == -1) {

perror("Error deleting file");

return -1;

}

std::cout << "File deleted successfully: " << fileName << std::endl;

return 0;

}

int main() {

const char \*fileName = "example.txt";

const char \*initialData = "Hello, this is a test.\n";

const char \*appendData = "Appending more data.\n";

if (createFile(fileName) == -1) return 1;

if (writeFile(fileName, initialData) == -1) return 1;

if (readFile(fileName) == -1) return 1;

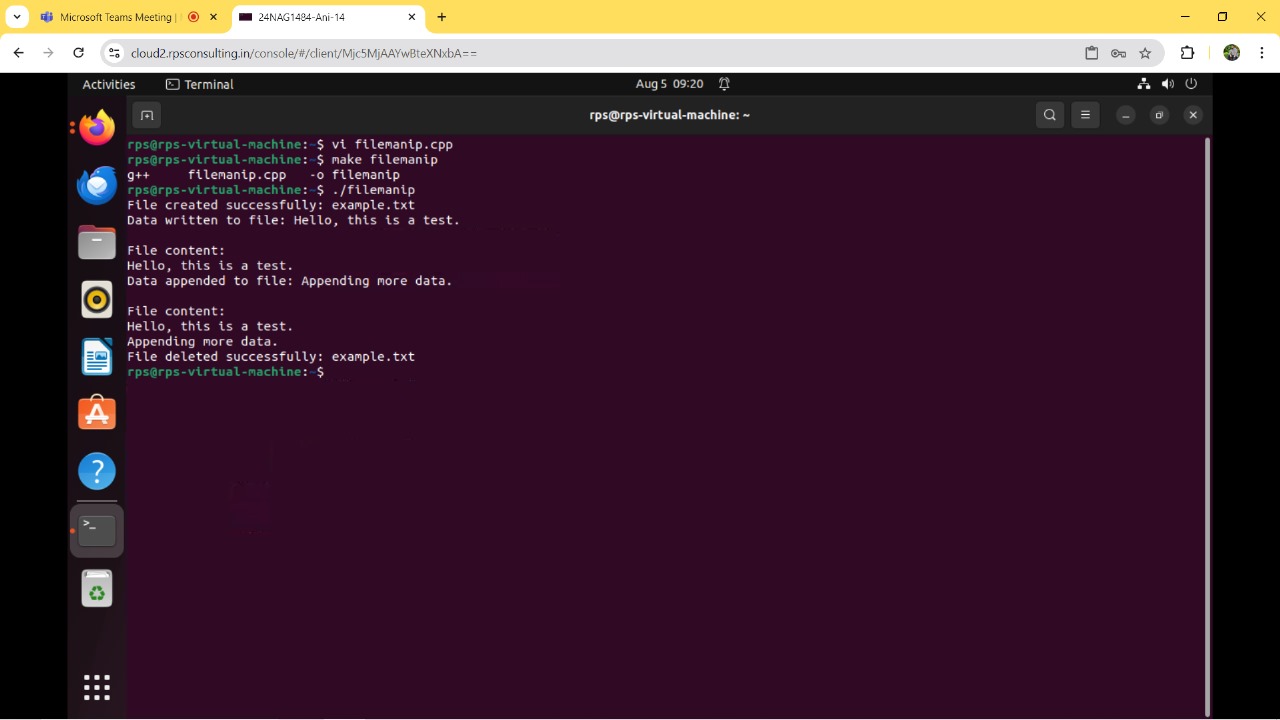
if (appendFile(fileName, appendData) == -1) return 1;

if (readFile(fileName) == -1) return 1;

if (deleteFile(fileName) == -1) return 1;

return 0;

}



Example :

Server:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define DEFAULT\_PORT 8080

#define DEFAULT\_BUFLEN 512

int main() {

int serverSocket;

struct sockaddr\_in serverAddr, clientAddr;

socklen\_t clientAddrLen = sizeof(clientAddr);

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

// Create a socket for the server

serverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (serverSocket < 0) {

std::cerr << "Socket creation failed" << std::endl;

return 1;

}

// Set up the sockaddr\_in structure

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(DEFAULT\_PORT);

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

// Bind the socket

if (bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Bind failed" << std::endl;

close(serverSocket);

return 1;

}

// Receive data

while (true) {

int recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr\*)&clientAddr, &clientAddrLen);

if (recvLen < 0) {

std::cerr << "recvfrom failed" << std::endl;

close(serverSocket);

return 1;

}

recvbuf[recvLen] = '\0'; // Null-terminate the received data

std::cout << "Received: " << recvbuf << std::endl;

// Echo the data back to the client

int sendLen = sendto(serverSocket, recvbuf, recvLen, 0, (struct sockaddr\*)&clientAddr, clientAddrLen);

if (sendLen < 0) {

std::cerr << "sendto failed" << std::endl;

close(serverSocket);

return 1;

}

}

// Cleanup

close(serverSocket);

return 0;

}

Clint :

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define DEFAULT\_PORT 8080

#define DEFAULT\_BUFLEN 512

int main() {

int clientSocket;

struct sockaddr\_in serverAddr;

char sendbuf[DEFAULT\_BUFLEN] = "Hello from client";

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

socklen\_t serverAddrLen = sizeof(serverAddr);

// Create a socket for the client

clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (clientSocket < 0) {

std::cerr << "Socket creation failed" << std::endl;

return 1;

}

// Set up the sockaddr\_in structure

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(DEFAULT\_PORT);

inet\_pton(AF\_INET, "127.0.0.1", &serverAddr.sin\_addr);

// Send data to the server

int sendLen = sendto(clientSocket, sendbuf, strlen(sendbuf), 0, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

if (sendLen < 0) {

std::cerr << "sendto failed" << std::endl;

close(clientSocket);

return 1;

}

std::cout << "Sent: " << sendbuf << std::endl;

// Receive data from the server

int recvLen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, (struct sockaddr\*)&serverAddr, &serverAddrLen);

if (recvLen < 0) {

std::cerr << "recvfrom failed" << std::endl;

close(clientSocket);

return 1;

}

recvbuf[recvLen] = '\0'; // Null-terminate the received data

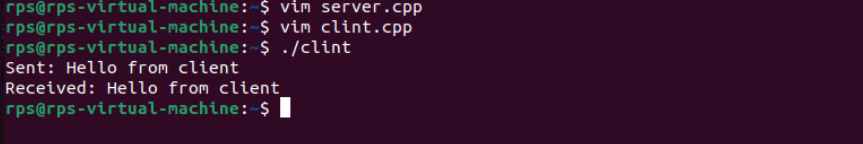
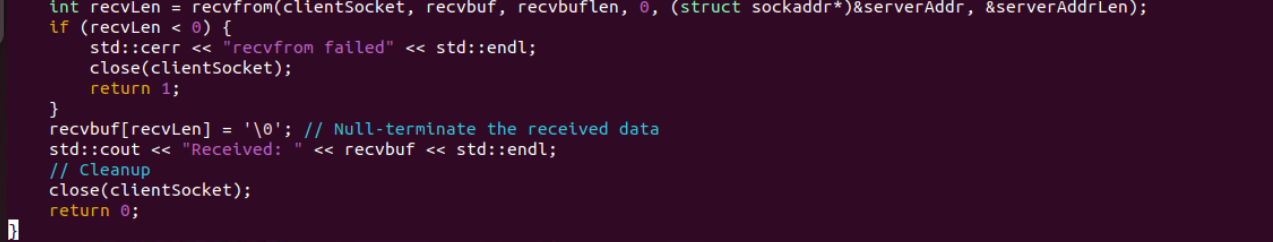
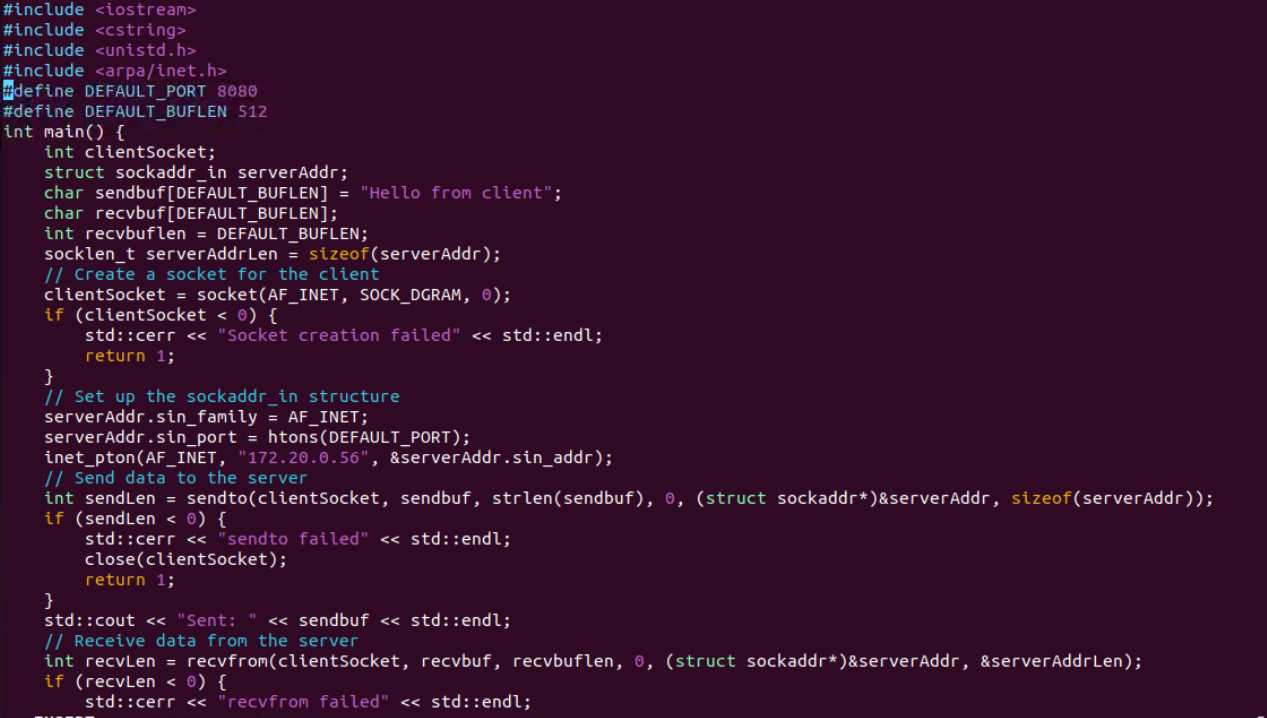
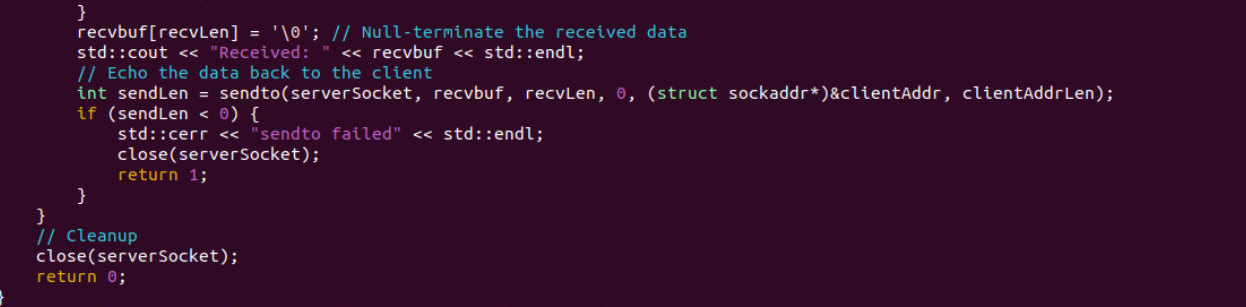
std::cout << "Received: " << recvbuf << std::endl;

// Cleanup

close(clientSocket);

return 0;

}



Example :

Server: #include <iostream>

#include <fstream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define DEFAULT\_PORT 8080

#define DEFAULT\_BUFLEN 512

int main() {

int serverSocket;

struct sockaddr\_in serverAddr, clientAddr;

socklen\_t clientAddrLen = sizeof(clientAddr);

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

// Create a socket for the server

serverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (serverSocket < 0) {

std::cerr << "Socket creation failed" << std::endl;

return 1;

}

// Set up the sockaddr\_in structure

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(DEFAULT\_PORT);

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

// Bind the socket

if (bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

std::cerr << "Bind failed" << std::endl;

close(serverSocket);

return 1;

}

// Receive file type

int recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr\*)&clientAddr, &clientAddrLen);

if (recvLen < 0) {

std::cerr << "recvfrom failed" << std::endl;

close(serverSocket);

return 1;

}

recvbuf[recvLen] = '\0'; // Null-terminate the received data

std::string fileType(recvbuf);

std::cout << "Received file type: " << fileType << std::endl;

// Receive file content

recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr\*)&clientAddr, &clientAddrLen);

if (recvLen < 0) {

std::cerr << "recvfrom failed" << std::endl;

close(serverSocket);

return 1;

}

recvbuf[recvLen] = '\0'; // Null-terminate the received data

std::string fileContent(recvbuf);

std::cout << "Received file content: " << fileContent << std::endl;

// Recreate the file

std::ofstream outFile("received\_file." + fileType, std::ios::out | std::ios::binary);

if (!outFile.is\_open()) {

std::cerr << "Failed to open file for writing" << std::endl;

close(serverSocket);

return 1;

}

outFile << fileContent;

outFile.close();

std::cout << "File recreated: received\_file." << fileType << std::endl;

// Cleanup

close(serverSocket);

return 0;

}

Clint :

#include <iostream>

#include <fstream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define DEFAULT\_PORT 8080

#define DEFAULT\_BUFLEN 512

int main() {

int clientSocket;

struct sockaddr\_in serverAddr;

socklen\_t serverAddrLen = sizeof(serverAddr);

// Read the file and type

std::ifstream file("1.txt", std::ios::in | std::ios::binary);

if (!file.is\_open()) {

std::cerr << "Failed to open file" << std::endl;

return 1;

}

std::string fileType = "txt";

std::string fileContent((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

file.close();

// Create a socket for the client

clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (clientSocket < 0) {

std::cerr << "Socket creation failed" << std::endl;

return 1;

}

// Set up the sockaddr\_in structure

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(DEFAULT\_PORT);

inet\_pton(AF\_INET, "127.0.0.1", &serverAddr.sin\_addr);

// Send file type

int sendLen = sendto(clientSocket, fileType.c\_str(), fileType.size(), 0, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

if (sendLen < 0) {

std::cerr << "sendto failed" << std::endl;

close(clientSocket);

return 1;

}

// Send file content

sendLen = sendto(clientSocket, fileContent.c\_str(), fileContent.size(), 0, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

if (sendLen < 0) {

std::cerr << "sendto failed" << std::endl;

close(clientSocket);

return 1;

}

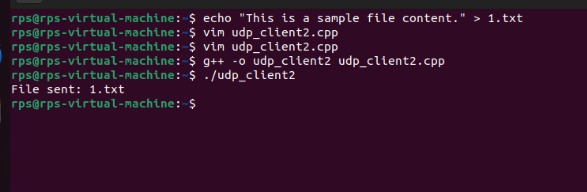
std::cout << "File sent: 1.txt" << std::endl;

// Cleanup

close(clientSocket);

return 0;

}



Example :

Server :

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 2020

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_len;

ssize\_t received\_len;

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

std::cerr << "Failed to create socket" << std::endl;

return 1;

}

memset(&server\_addr, 0, sizeof(server\_addr));

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

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

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

if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

std::cerr << "Failed to bind socket" << std::endl;

close(sockfd);

return 1;

}

std::cout << "UDP server listening on port " << PORT << std::endl;

while (true) {

memset(&client\_addr, 0, sizeof(client\_addr));

client\_len = sizeof(client\_addr);

received\_len = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&client\_addr, &client\_len);

if (received\_len < 0) {

std::cerr << "Failed to receive message" << std::endl;

continue;

}

buffer[received\_len] = '\0';

std::cout << "Received message: " << buffer << std::endl;

std::cout << "From client: " << inet\_ntoa(client\_addr.sin\_addr) << ":" << ntohs(client\_addr.sin\_port) << std::endl;

const char \*ack\_msg = "ACK";

if (sendto(sockfd, ack\_msg, strlen(ack\_msg), 0, (struct sockaddr \*)&client\_addr, client\_len) < 0) {

std::cerr << "Failed to send acknowledgment" << std::endl;

}

}

close(sockfd);

return 0;

}

Clint:

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <chrono>

#include <thread>

#define SERVER\_PORT 2020

#define BUFFER\_SIZE 1024

#define TIMEOUT\_MS 1000

#define MAX\_RETRIES 5

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

std::string message;

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

std::cerr << "Failed to create socket" << std::endl;

return 1;

}

memset(&server\_addr, 0, sizeof(server\_addr));

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

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(SERVER\_PORT);

while (true) {

std::cout << "Enter message to send: ";

std::getline(std::cin, message);

for (int attempt = 0; attempt < MAX\_RETRIES; ++attempt) {

if (sendto(sockfd, message.c\_str(), message.length(), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

std::cerr << "Failed to send message" << std::endl;

continue;

}

struct timeval tv;

tv.tv\_sec = TIMEOUT\_MS / 1000;

tv.tv\_usec = (TIMEOUT\_MS % 1000) \* 1000;

setsockopt(sockfd, SOL\_SOCKET, SO\_RCVTIMEO, &tv, sizeof(tv));

socklen\_t server\_len = sizeof(server\_addr);

ssize\_t received\_len = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&server\_addr, &server\_len);

if (received\_len >= 0) {

buffer[received\_len] = '\0';

if (strcmp(buffer, "ACK") == 0) {

std::cout << "Server response: " << buffer << std::endl;

break;

}

} else {

std::cerr << "No acknowledgment received, retrying..." << std::endl;

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(TIMEOUT\_MS));

}

}

close(sockfd);

return 0;

}

