DATE :- 05-08-2024

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).

```
rps@rps-virtual-machine:~/amee$ vim file.cpp
rps@rps-virtual-machine:~/amee$ g++ file.cpp -o file
rps@rps-virtual-machine:~/amee$ ./file
File created successfully.
Content written successfully.
Reading file contents:
Hello, World!
Content appended successfully.

Reading file contents after appending:
Hello, World!
This is additional text.
File deleted successfully.
rps@rps-virtual-machine:~/amee$
```

```cpp
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <cstdio>

// Function to create a new file
void createFile(const char* filename) {
    int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        perror("Error creating file");
        return;
    }
    close(fd);
    std::cout << "File created successfully." << std::endl;
}

// Function to write a string to the file
void writeFile(const char* filename, const char* content) {
    int fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        perror("Error opening file for writing");
        return;
    }
    if (write(fd, content, strlen(content)) == -1) {
        perror("Error writing to file");
    } else {
        std::cout << "Content written successfully." << std::endl;
    }
    close(fd);
}

// Function to read the contents of the file
void readFile(const char* filename) {
    int fd = open(filename, O_RDONLY);
    if (fd == -1) {
```

```
    if (fd == -1) {
        perror("Error opening file for reading");
        return;
    }
    char buffer[1024];
    ssize_t bytesRead;
    while ((bytesRead = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0';
        std::cout << buffer;
    }
    if (bytesRead == -1) {
        perror("Error reading file");
    }
    close(fd);
}

// Function to append additional text to the file
void appendToFile(const char* filename, const char* content) {
    int fd = open(filename, O_WRONLY | O_APPEND);
    if (fd == -1) {
        perror("Error opening file for appending");
        return;
    }
    if (write(fd, content, strlen(content)) == -1) {
        perror("Error appending to file");
    } else {
        std::cout << "Content appended successfully." << std::endl;
    }
    close(fd);
}

// Function to delete the file
void deleteFile(const char* filename) {
    if (unlink(filename) == -1) {
        perror("Error deleting file");
```

```
void deleteFile(const char* filename) {
    if (unlink(filename) == -1) {
        perror("Error deleting file");
    } else {
        std::cout << "File deleted successfully." << std::endl;
    }
}

int main() {
    const char* filename = "example.txt";
    const char* initialContent = "Hello, World!\n";
    const char* additionalContent = "This is additional text.\n";

    // Create a new file
    createFile(filename);

    // Write to the file
    writeFile(filename, initialContent);

    // Read the file
    std::cout << "Reading file contents:" << std::endl;
    readFile(filename);

    // Append to the file
    appendToFile(filename, additionalContent);

    // Read the file again
    std::cout << "\nReading file contents after appending:" << std::endl;
    readFile(filename);

    // Delete the file
    deleteFile(filename);

    return 0;
}
```

SERVER CODE :-

```
rps@rps-virtual-machine:~/amee$ vim server.cpp
rps@rps-virtual-machine:~/amee$ make server
g++     server.cpp   -o server
rps@rps-virtual-machine:~/amee$ ./server
Received: Hello from client
hii ameesha
Received: Hello from client
^C
rps@rps-virtual-machine:~/amee$
```

```cpp
#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
```

```cpp
        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;
}
```

CLIENT CODE :-

```
rps@rps-virtual-machine:~/amee$ vim client.cpp
rps@rps-virtual-machine:~/amee$ make client
g++      client.cpp    -o client
rps@rps-virtual-machine:~/amee$ ./client
Sent: Hello from client
Received: Hello from client
rps@rps-virtual-machine:~/amee$ ./client
Sent: Hello from client
Received: Hello from client
rps@rps-virtual-machine:~/amee$
```

```cpp
#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;
    }
```

```cpp
    // 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;
}
```

CODE FOR 1.TXT :-

SERVER CODE :-

```
rps@rps-virtual-machine:~/amee$ touch 1.txt
rps@rps-virtual-machine:~/amee$ cat 1.txt
rps@rps-virtual-machine:~/amee$ vi 1.txt
rps@rps-virtual-machine:~/amee$ vi 1.txt
rps@rps-virtual-machine:~/amee$ cat 1.txt
Hii this is ameesha jeruganti
Let me besties........
rps@rps-virtual-machine:~/amee$ vim server.cpp
rps@rps-virtual-machine:~/amee$ vim server1.cpp
rps@rps-virtual-machine:~/amee$ make server1
g++     server1.cpp   -o server1
rps@rps-virtual-machine:~/amee$ ./server
Received: txt
Received: Hii this is ameesha jeruganti
Let me besties........
```

```cpp
#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;
    }
```

```
        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;
}
```

CLIENT CODE :-

```
rps@rps-virtual-machine:~/amee$ vim client1.cpp
rps@rps-virtual-machine:~/amee$ make client1
g++     client1.cpp   -o client1
rps@rps-virtual-machine:~/amee$ ./client1
File sent: 1.txt
rps@rps-virtual-machine:~/amee$
```

```cpp
#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);
```

```cpp
    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;
}
```

UDP Server Implementation:

Create a UDP socket.

Bind the socket to a specified port.

Implement a loop to continuously listen for incoming messages.

Upon receiving a message:

Print the received message along with the client's address and port.

Send an acknowledgment message ("Message received") back to the client.

Ensure proper error handling and resource cleanup.

```
rps@rps-virtual-machine:~$ vim udpserver.cpp
rps@rps-virtual-machine:~$ make udpserver
g++      udpserver.cpp    -o udpserver
rps@rps-virtual-machine:~$ ./udpserver
Server is listening on port 12345
Received message: Hello, Server!
Client address: 127.0.0.1
Client port: 50065
^C
```

```
    std::cout << "Server is listening on port " << PORT << std::endl;

    while (true) {
        // Receive message
        ssize_t recvLen = recvfrom(sockfd, buffer, BUFFER_SIZE - 1, 0, (struct sockaddr*)&clientAddr, &addrLen);
        if (recvLen < 0) {
            std::cerr << "Receive failed" << std::endl;
            continue;
        }
        buffer[recvLen] = '\0'; // Null-terminate the received data

        // Print received message and client info
        std::cout << "Received message: " << buffer << std::endl;
        std::cout << "Client address: " << inet_ntoa(clientAddr.sin_addr) << std::endl;
        std::cout << "Client port: " << ntohs(clientAddr.sin_port) << std::endl;

        // Send acknowledgment
        if (sendto(sockfd, ackMessage, strlen(ackMessage), 0, (struct sockaddr*)&clientAddr, addrLen) < 0) {
            std::cerr << "Send failed" << std::endl;
        }
    }

    close(sockfd);
    return 0;
}
rps@rps-virtual-machine:~$
```

2. UDP Client Implementation:

Create a UDP socket.

Allow the user to input the server's IP address and port number.

Send a predefined message (e.g., "Hello, Server!") to the server.

Wait for an acknowledgment from the server.

Print the acknowledgment message to the console.

Ensure proper error handling and resource cleanup.

```
rps@rps-virtual-machine:~$ vim udpcilent.cpp
rps@rps-virtual-machine:~$ make udpcilent
g++      udpcilent.cpp    -o udpcilent
rps@rps-virtual-machine:~$ ./udpcilent
Enter server IP address: 127.0.0.1
Enter server port number: 12345
Acknowledgment from server: Message received
rps@rps-virtual-machine:~$
```

```
rps@rps-virtual-machine:~$ cat udpcilent.cpp
#include <iostream>
#include <cstring>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in serverAddr;
    char buffer[BUFFER_SIZE];
    std::string serverIp;
    int serverPort;
    const char *message = "Hello, Server!";

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Get server IP and port from user
    std::cout << "Enter server IP address: ";
    std::cin >> serverIp;
    std::cout << "Enter server port number: ";
    std::cin >> serverPort;

    // Setup server address
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(serverPort);
```

```
    if (inet_pton(AF_INET, serverIp.c_str(), &serverAddr.sin_addr) <= 0) {
        std::cerr << "Invalid IP address" << std::endl;
        close(sockfd);
        return 1;
    }

    // Send message
    if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Send failed" << std::endl;
        close(sockfd);
        return 1;
    }

    // Receive acknowledgment
    socklen_t addrLen = sizeof(serverAddr);
    ssize_t recvLen = recvfrom(sockfd, buffer, BUFFER_SIZE - 1, 0, (struct sockaddr*)&serverAddr, &addrLen);
    if (recvLen < 0) {
        std::cerr << "Receive failed" << std::endl;
        close(sockfd);
        return 1;
    }
    buffer[recvLen] = '\0'; // Null-terminate the received data

    // Print acknowledgment
    std::cout << "Acknowledgment from server: " << buffer << std::endl;

    close(sockfd);
    return 0;
}
rps@rps-virtual-machine:~$
```