```
#include <iostream>
#include <string>
#include <cstring>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

const int PORT = 8080;
const int BUFFER_SIZE = 1024;
const char XOR_KEY = 0xAA;  // Simple XOR encryption key

void xor_encrypt_decrypt(char *data, size_t length) {
    for (size_t i = 0; i < length; ++i) {
        data[i] ^= XOR_KEY;
    }
}

int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "SERVER_IP_ADDRESS", &server_address.sin_addr) <= 0) {
        std::cerr << "Invalid address or address not supported" << std::endl;
        close(client_socket);
        return 1;
    }

    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        std::cerr << "Connection failed" << std::endl;
        close(client_socket);
        return 1;
    }

    std::string message;
    char buffer[BUFFER_SIZE];

    while (true) {
        std::cout << "Enter message: ";
        std::getline(std::cin, message);

        if (message == "exit") {
            break;
        }

        // Encrypt message
        xor_encrypt_decrypt(&message[0], message.size());

        send(client_socket, message.c_str(), message.size(), 0);

        memset(buffer, 0, BUFFER_SIZE);
        ssize_t bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            std::cerr << "Server disconnected or error occurred" << std::endl;
            break;
        }

        // Decrypt received data
        xor_encrypt_decrypt(buffer, bytes_received);

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

    close(client_socket);
    return 0;
}


#include <iostream>
#include <string>
#include <cstring>
#include <thread>
#include <vector>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

const int PORT = 8080;
const int BUFFER_SIZE = 1024;
const char XOR_KEY = 0xAA;  // Simple XOR encryption key

void xor_encrypt_decrypt(char *data, size_t length) {
    for (size_t i = 0; i < length; ++i) {
        data[i] ^= XOR_KEY;
    }
}

void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE];
    while (true) {
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            std::cerr << "Client disconnected or error occurred" << std::endl;
            close(client_socket);
            return;
        }

        // Decrypt received data
        xor_encrypt_decrypt(buffer, bytes_received);

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

        // Echo back the received data (encrypted)
        xor_encrypt_decrypt(buffer, bytes_received);
        send(client_socket, buffer, bytes_received, 0);
    }
}

int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_socket, (struct sockaddr *)&address, sizeof(address)) < 0) {
        std::cerr << "Binding failed" << std::endl;
        close(server_socket);
        return 1;
    }
    if (listen(server_socket, 3) < 0) {
        std::cerr << "Listening failed" << std::endl;
        close(server_socket);
        return 1;
    }

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

    while (true) {
        int client_socket = accept(server_socket, NULL, NULL);
        if (client_socket < 0) {
            std::cerr << "Client acceptance failed" << std::endl;
            close(server_socket);
            return 1;
        }

        std::thread client_thread(handle_client, client_socket);
        client_thread.detach();
    }

    close(server_socket);
    return 0;
}
```

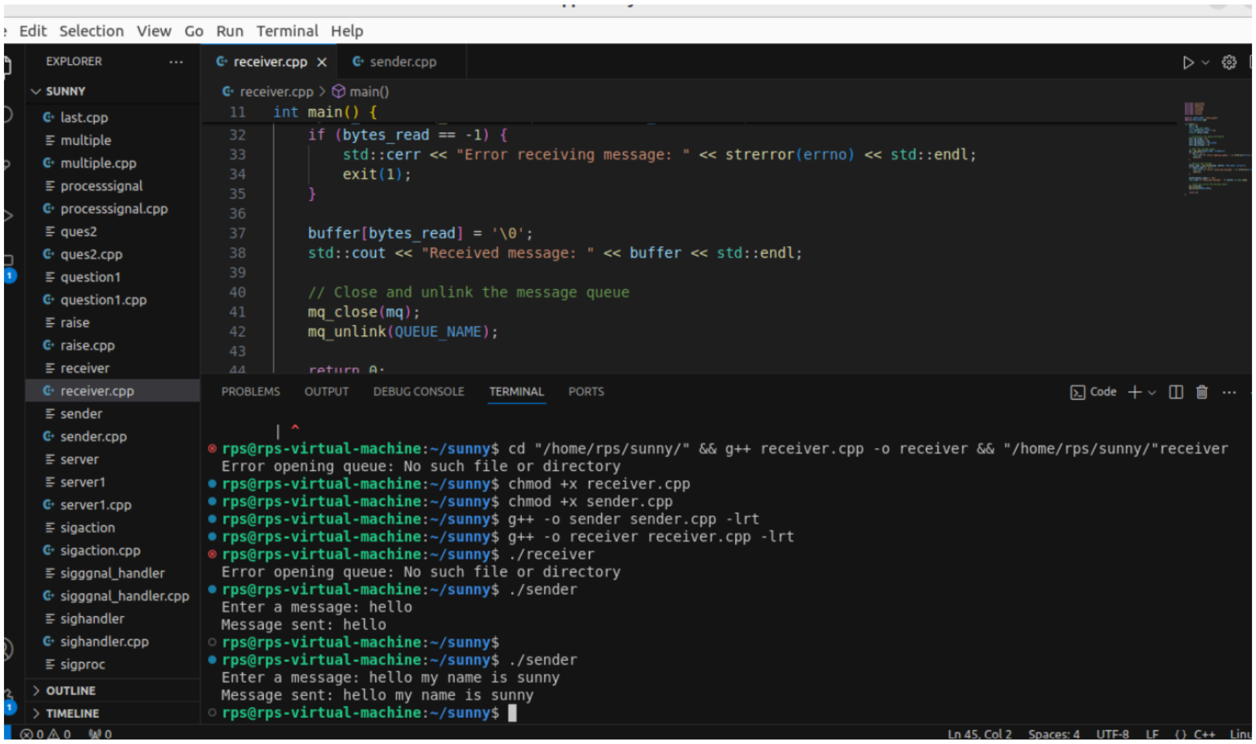
```
#include <iostream>
#include <mqueue.h>
#include <cstring>
#include <cstdlib>
#include <cerrno>
#include <cstdio>

#define QUEUE_NAME "/test_queue"
#define MAX_SIZE 1024
#define MSG_STOP "exit"

int main() {
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE];

    // Initialize the queue attributes
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    // Create the message queue
    mq = mq_open(QUEUE_NAME, O_CREAT | O_WRONLY, 0644, &attr);
    if (mq == -1) {
        std::cerr << "Error creating queue: " << strerror(errno) << std::endl;
        exit(1);
    }

    std::cout << "Enter a message: ";
    std::cin.getline(buffer, MAX_SIZE);

    // Send the message
    if (mq_send(mq, buffer, strlen(buffer) + 1, 0) == -1) {
        std::cerr << "Error sending message: " << strerror(errno) << std::endl;
        exit(1);
    }

    std::cout << "Message sent: " << buffer << std::endl;

    // Close the message queue
    mq_close(mq);

    return 0;
}
```

```
g++ -o sender sender.cpp -lrt
```

```
#include <iostream>
#include <mqueue.h>
#include <cstring>
#include <cstdlib>
```

```
#include <cerrno>
#include <cstdio>

#define QUEUE_NAME "/test_queue"
#define MAX_SIZE 1024

int main() {
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE + 1];
    ssize_t bytes_read;

    // Initialize the queue attributes
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    // Open the message queue
    mq = mq_open(QUEUE_NAME, O_RDONLY);
    if (mq == -1) {
        std::cerr << "Error opening queue: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Receive the message
    bytes_read = mq_receive(mq, buffer, MAX_SIZE, nullptr);
    if (bytes_read == -1) {
        std::cerr << "Error receiving message: " << strerror(errno) << std::endl;
        exit(1);
    }

    buffer[bytes_read] = '\0';
    std::cout << "Received message: " << buffer << std::endl;

    // Close and unlink the message queue
    mq_close(mq);
    mq_unlink(QUEUE_NAME);

    return 0;
}
```

g++ -o receiver receiver.cpp -lrt

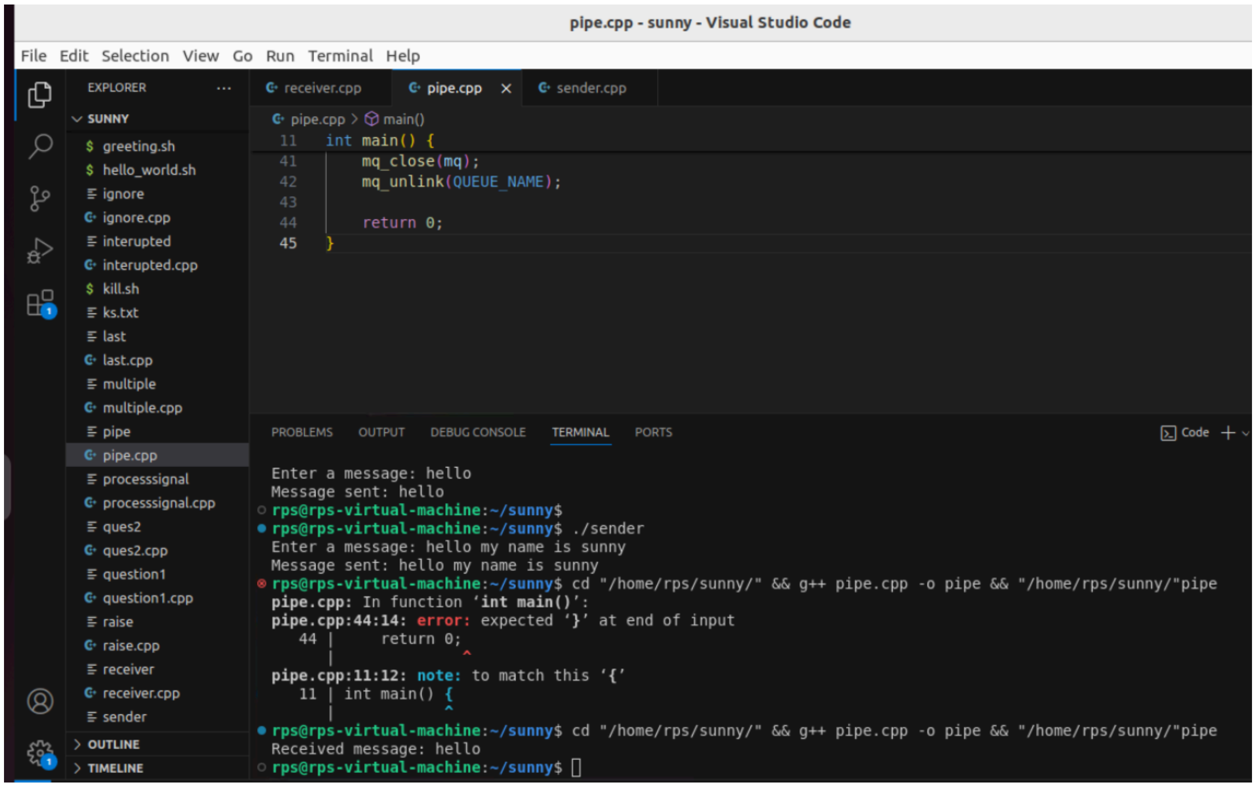


```cpp
int main() {
    if (bytes_read == -1) {
        std::cerr << "Error receiving message: " << strerror(errno) << std::endl;
        exit(1);
    }

    buffer[bytes_read] = '\0';
    std::cout << "Received message: " << buffer << std::endl;

    // Close and unlink the message queue
    mq_close(mq);
    mq_unlink(QUEUE_NAME);

    return 0;
```

```
      | ^
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ receiver.cpp -o receiver && "/home/rps/sunny/"receiver
Error opening queue: No such file or directory
rps@rps-virtual-machine:~/sunny$ chmod +x receiver.cpp
rps@rps-virtual-machine:~/sunny$ chmod +x sender.cpp
rps@rps-virtual-machine:~/sunny$ g++ -o sender sender.cpp -lrt
rps@rps-virtual-machine:~/sunny$ g++ -o receiver receiver.cpp -lrt
rps@rps-virtual-machine:~/sunny$ ./receiver
Error opening queue: No such file or directory
rps@rps-virtual-machine:~/sunny$ ./sender
Enter a message: hello
Message sent: hello
rps@rps-virtual-machine:~/sunny$
rps@rps-virtual-machine:~/sunny$ ./sender
Enter a message: hello my name is sunny
Message sent: hello my name is sunny
rps@rps-virtual-machine:~/sunny$
```

PIPE

```cpp
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <fcntl.h>
#include <sys/stat.h>
#include <cerrno>

#define PIPE1 "/tmp/pipe1"
#define PIPE2 "/tmp/pipe2"
#define MAX_SIZE 1024

void create_pipe(const char* pipe_name) {
    if (mkfifo(pipe_name, 0666) == -1) {
        if (errno != EEXIST) {
            std::cerr << "Error creating pipe " << pipe_name << ": " << strerror(errno) << std::endl;
            exit(1);
        }
    }
}

int main() {
    // Create the pipes
    create_pipe(PIPE1);
    create_pipe(PIPE2);

    int pipe1_fd, pipe2_fd;
    char buffer[MAX_SIZE];

    while (true) {
        // Read message from PIPE1
        pipe1_fd = open(PIPE1, O_RDONLY);
        if (pipe1_fd == -1) {
            std::cerr << "Error opening pipe1 for reading: " << strerror(errno) << std::endl;
            exit(1);
        }
        read(pipe1_fd, buffer, MAX_SIZE);
        std::cout << "Process2 received: " << buffer << std::endl;
        close(pipe1_fd);

        if (strcmp(buffer, "exit") == 0) break;

        // Write message to PIPE2
        std::cout << "Process2, enter a message: ";
        std::cin.getline(buffer, MAX_SIZE);
        pipe2_fd = open(PIPE2, O_WRONLY);
        if (pipe2_fd == -1) {
            std::cerr << "Error opening pipe2 for writing: " << strerror(errno) << std
```

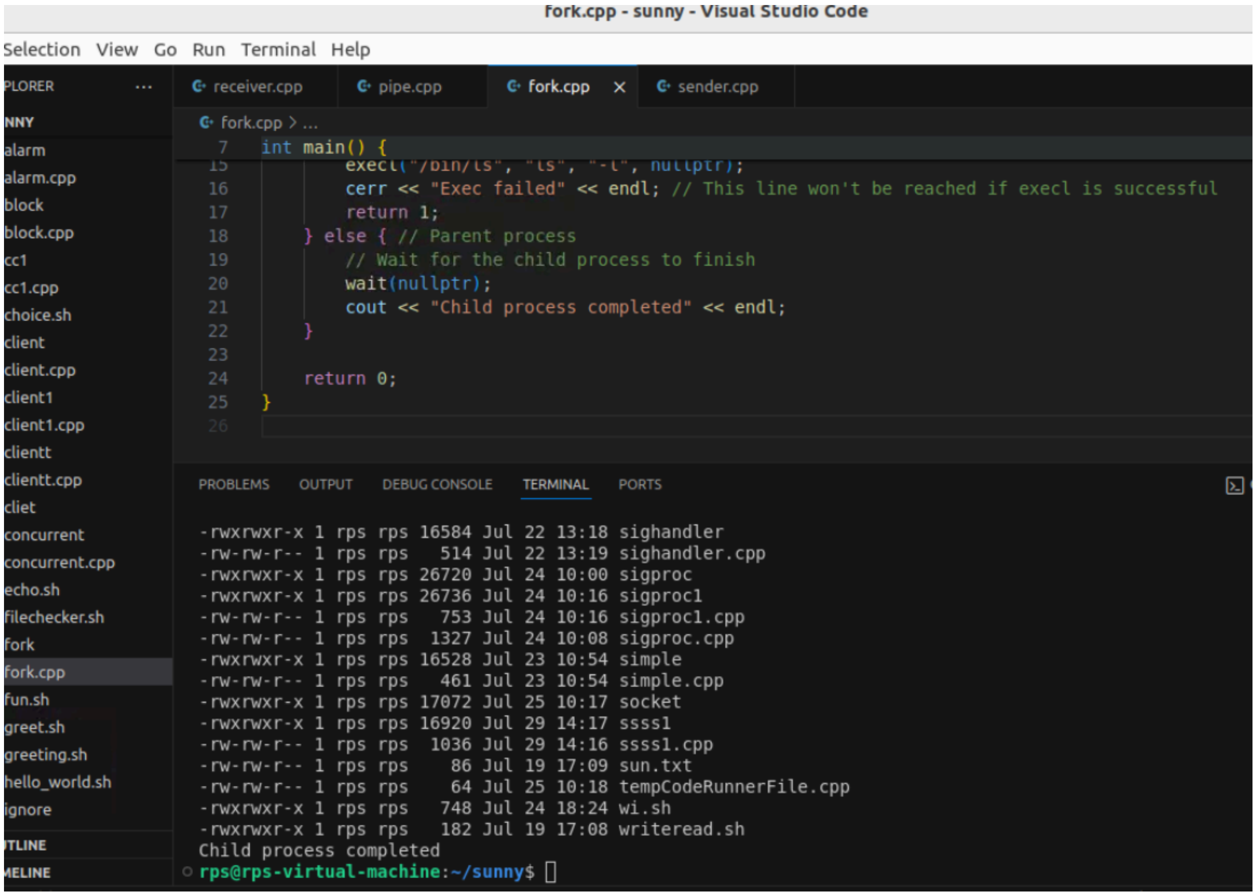


```
int main() {
    mq_close(mq);
    mq_unlink(QUEUE_NAME);

    return 0;
}
```

```
Enter a message: hello
Message sent: hello
rps@rps-virtual-machine:~/sunny$
rps@rps-virtual-machine:~/sunny$ ./sender
Enter a message: hello my name is sunny
Message sent: hello my name is sunny
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ pipe.cpp -o pipe && "/home/rps/sunny/"pipe
pipe.cpp: In function 'int main()':
pipe.cpp:44:14: error: expected '}' at end of input
   44 |     return 0;
      |              ^
pipe.cpp:11:12: note: to match this '{'
   11 | int main() {
      |            ^
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ pipe.cpp -o pipe && "/home/rps/sunny/"pipe
Received message: hello
rps@rps-virtual-machine:~/sunny$
```

FORK

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        cerr << "Fork failed" << endl;
        return 1;
    } else if (pid == 0) { // Child process
        // Replace the current process with the "ls" command
        execl("/bin/ls", "ls", "-l", nullptr);
        cerr << "Exec failed" << endl; // This line won't be reached if execl is successful
        return 1;
    } else { // Parent process
        // Wait for the child process to finish
        wait(nullptr);
        cout << "Child process completed" << endl;
    }

    return 0;
}
```

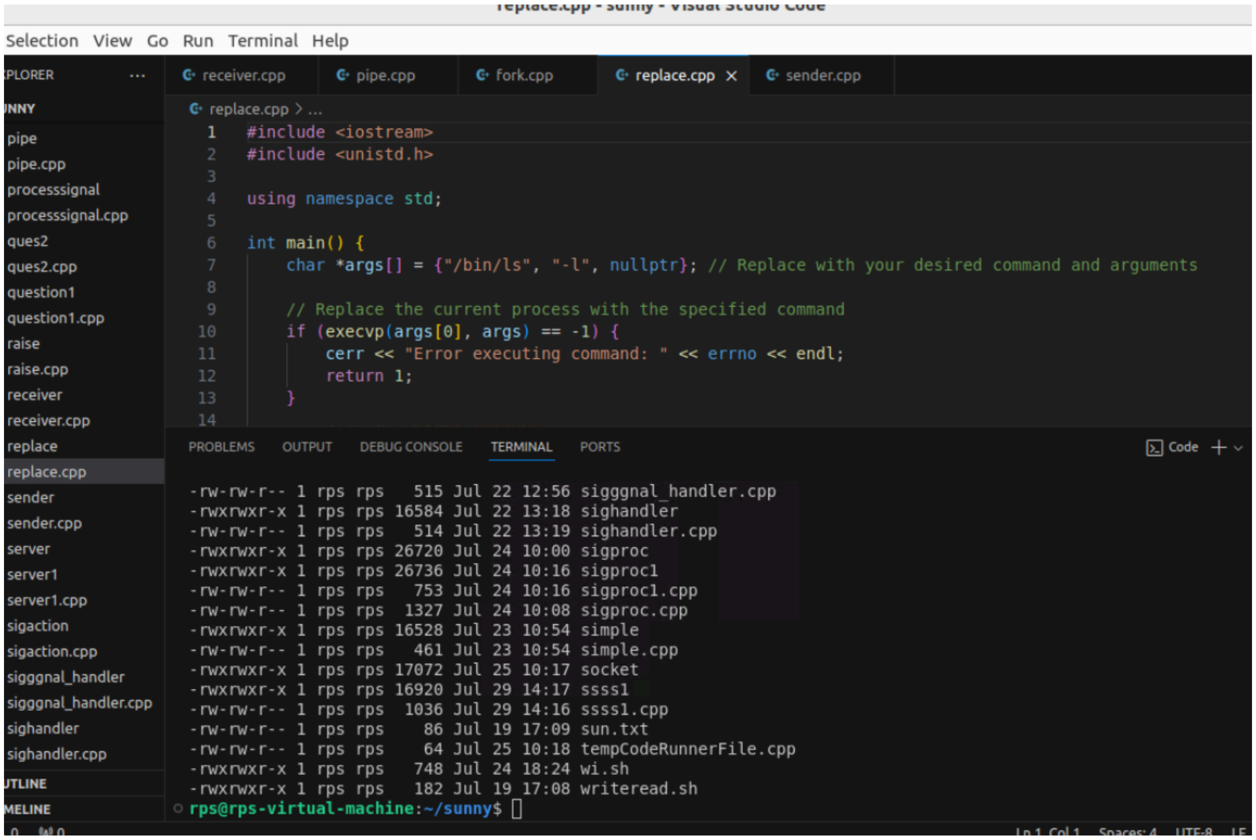


```
int main() {
        execl("/bin/ls", "ls", "-l", nullptr);
        cerr << "Exec failed" << endl; // This line won't be reached if execl is successful
        return 1;
    } else { // Parent process
        // Wait for the child process to finish
        wait(nullptr);
        cout << "Child process completed" << endl;
    }

    return 0;
}
```

```
-rwxrwxr-x 1 rps rps 16584 Jul 22 13:18 sighandler
-rw-rw-r-- 1 rps rps   514 Jul 22 13:19 sighandler.cpp
-rwxrwxr-x 1 rps rps 26720 Jul 24 10:00 sigproc
-rwxrwxr-x 1 rps rps 26736 Jul 24 10:16 sigproc1
-rw-rw-r-- 1 rps rps   753 Jul 24 10:16 sigproc1.cpp
-rw-rw-r-- 1 rps rps  1327 Jul 24 10:08 sigproc.cpp
-rwxrwxr-x 1 rps rps 16528 Jul 23 10:54 simple
-rw-rw-r-- 1 rps rps   461 Jul 23 10:54 simple.cpp
-rwxrwxr-x 1 rps rps 17072 Jul 25 10:17 socket
-rwxrwxr-x 1 rps rps 16920 Jul 29 14:17 ssss1
-rw-rw-r-- 1 rps rps  1036 Jul 29 14:16 ssss1.cpp
-rw-rw-r-- 1 rps rps    86 Jul 19 17:09 sun.txt
-rw-rw-r-- 1 rps rps    64 Jul 25 10:18 tempCodeRunnerFile.cpp
-rwxrwxr-x 1 rps rps   748 Jul 24 18:24 wi.sh
-rwxrwxr-x 1 rps rps   182 Jul 19 17:08 writeread.sh
Child process completed
rps@rps-virtual-machine:~/sunny$
```

REPLACE

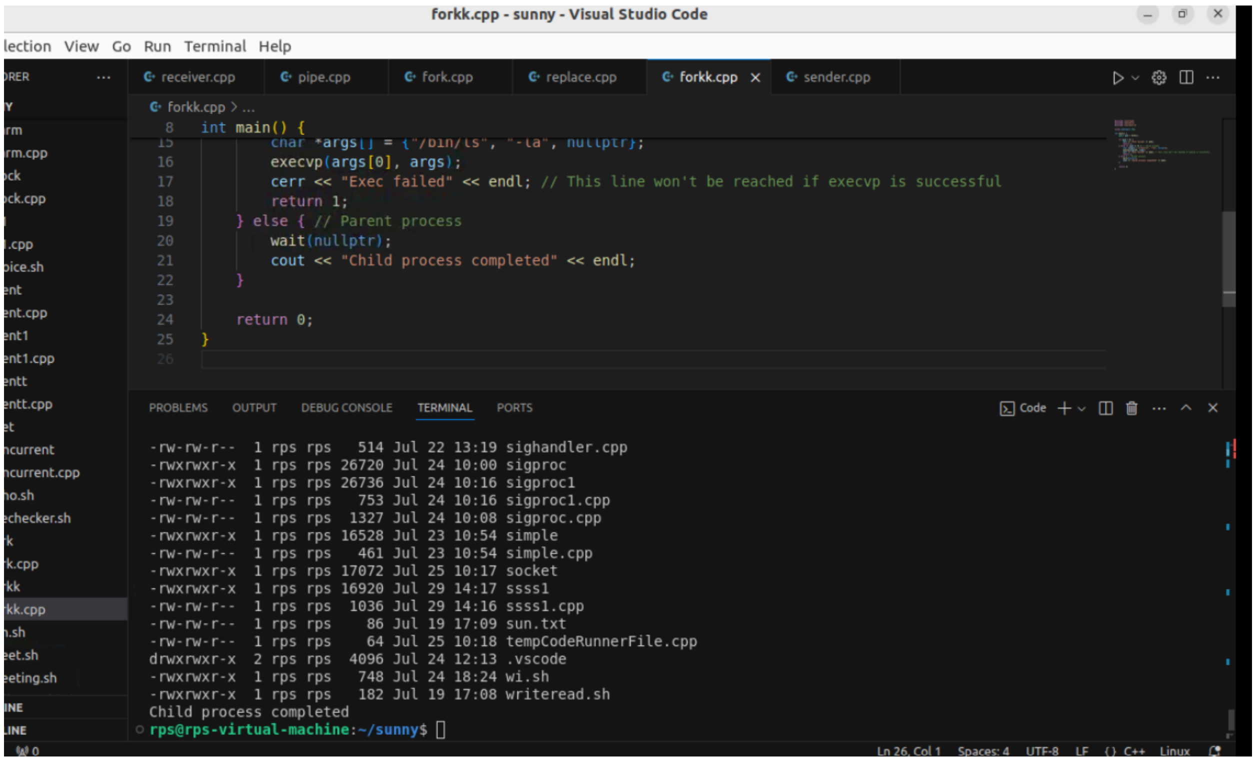
```
#include <iostream>
#include <unistd.h>

using namespace std;

int main() {
    char *args[] = {"/bin/ls", "-l", nullptr}; // Replace with your desired command and arguments

    // Replace the current process with the specified command
    if (execvp(args[0], args) == -1) {
        cerr << "Error executing command: " << errno << endl;
        return 1;
    }

    // This line will not be reached if execvp is successful
    cerr << "This should not be printed" << endl;
    return 0;
}
```



```
#include <iostream>
#include <unistd.h>

using namespace std;

int main() {
    char *args[] = {"/bin/ls", "-l", nullptr}; // Replace with your desired command and arguments

    // Replace the current process with the specified command
    if (execvp(args[0], args) == -1) {
        cerr << "Error executing command: " << errno << endl;
        return 1;
    }
```

```
-rw-rw-r-- 1 rps rps   515 Jul 22 12:56 sigggnal_handler.cpp
-rwxrwxr-x 1 rps rps 16584 Jul 22 13:18 sighandler
-rw-rw-r-- 1 rps rps   514 Jul 22 13:19 sighandler.cpp
-rwxrwxr-x 1 rps rps 26720 Jul 24 10:00 sigproc
-rwxrwxr-x 1 rps rps 26736 Jul 24 10:16 sigproc1
-rw-rw-r-- 1 rps rps   753 Jul 24 10:16 sigproc1.cpp
-rw-rw-r-- 1 rps rps  1327 Jul 24 10:08 sigproc.cpp
-rwxrwxr-x 1 rps rps 16528 Jul 23 10:54 simple
-rw-rw-r-- 1 rps rps   461 Jul 23 10:54 simple.cpp
-rwxrwxr-x 1 rps rps 17072 Jul 25 10:17 socket
-rwxrwxr-x 1 rps rps 16920 Jul 29 14:17 sssss1
-rw-rw-r-- 1 rps rps  1036 Jul 29 14:16 sssss1.cpp
-rw-rw-r-- 1 rps rps    86 Jul 19 17:09 sun.txt
-rw-rw-r-- 1 rps rps    64 Jul 25 10:18 tempCodeRunnerFile.cpp
-rwxrwxr-x 1 rps rps   748 Jul 24 18:24 wi.sh
-rwxrwxr-x 1 rps rps   182 Jul 19 17:08 writeread.sh
rps@rps-virtual-machine:~/sunny$
```

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        cerr << "Fork failed" << endl;
        return 1;
    } else if (pid == 0) { // Child process
        char *args[] = {"/bin/ls", "-la", nullptr};
        execvp(args[0], args);
        cerr << "Exec failed" << endl; // This line won't be reached if execvp is successful
        return 1;
    } else { // Parent process
        wait(nullptr);
        cout << "Child process completed" << endl;
    }

    return 0;
}
```


forkk.cpp - sunny - Visual Studio Code
lection View Go Run Terminal Help
receiver.cpp pipe.cpp fork.cpp replace.cpp forkk.cpp sender.cpp
forkk.cpp > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 26, Col 1 Spaces: 4 UTF-8 LF {} C++ Linux

```
int main() {
        char *args[] = {"/bin/ls", "-la", nullptr};
        execvp(args[0], args);
        cerr << "Exec failed" << endl; // This line won't be reached if execvp is successful
        return 1;
    } else { // Parent process
        wait(nullptr);
        cout << "Child process completed" << endl;
    }

    return 0;
}
```

```
-rw-rw-r--  1 rps rps   514 Jul 22 13:19 sighandler.cpp
-rwxrwxr-x  1 rps rps 26720 Jul 24 10:00 sigproc
-rwxrwxr-x  1 rps rps 26736 Jul 24 10:16 sigproc1
-rw-rw-r--  1 rps rps   753 Jul 24 10:16 sigproc1.cpp
-rw-rw-r--  1 rps rps  1327 Jul 24 10:08 sigproc.cpp
-rwxrwxr-x  1 rps rps 16528 Jul 23 10:54 simple
-rw-rw-r--  1 rps rps   461 Jul 23 10:54 simple.cpp
-rwxrwxr-x  1 rps rps 17072 Jul 25 10:17 socket
-rwxrwxr-x  1 rps rps 16920 Jul 29 14:17 ssss1
-rw-rw-r--  1 rps rps  1036 Jul 29 14:16 ssss1.cpp
-rw-rw-r--  1 rps rps    86 Jul 19 17:09 sun.txt
-rw-rw-r--  1 rps rps    64 Jul 25 10:18 tempCodeRunnerFile.cpp
drwxrwxr-x  2 rps rps  4096 Jul 24 12:13 .vscode
-rwxrwxr-x  1 rps rps   748 Jul 24 18:24 wi.sh
-rwxrwxr-x  1 rps rps   182 Jul 19 17:08 writeread.sh
Child process completed
rps@rps-virtual-machine:~/sunny$ 
```