Problem Statement: Inter-Process Communication (IPC) using Pipes, Shared Memory, and Message Queues

Design and implement efficient and reliable inter-process communication (IPC) mechanisms using pipes, shared memory, and message queues in C to facilitate data exchange and synchronization between multiple processes within a single system.

Specific Requirements:

Pipe: Create and manage unidirectional and bidirectional pipes for simple data transfer between related processes.

Shared Memory: Allocate and manage shared memory segments for efficient data sharing between multiple processes.

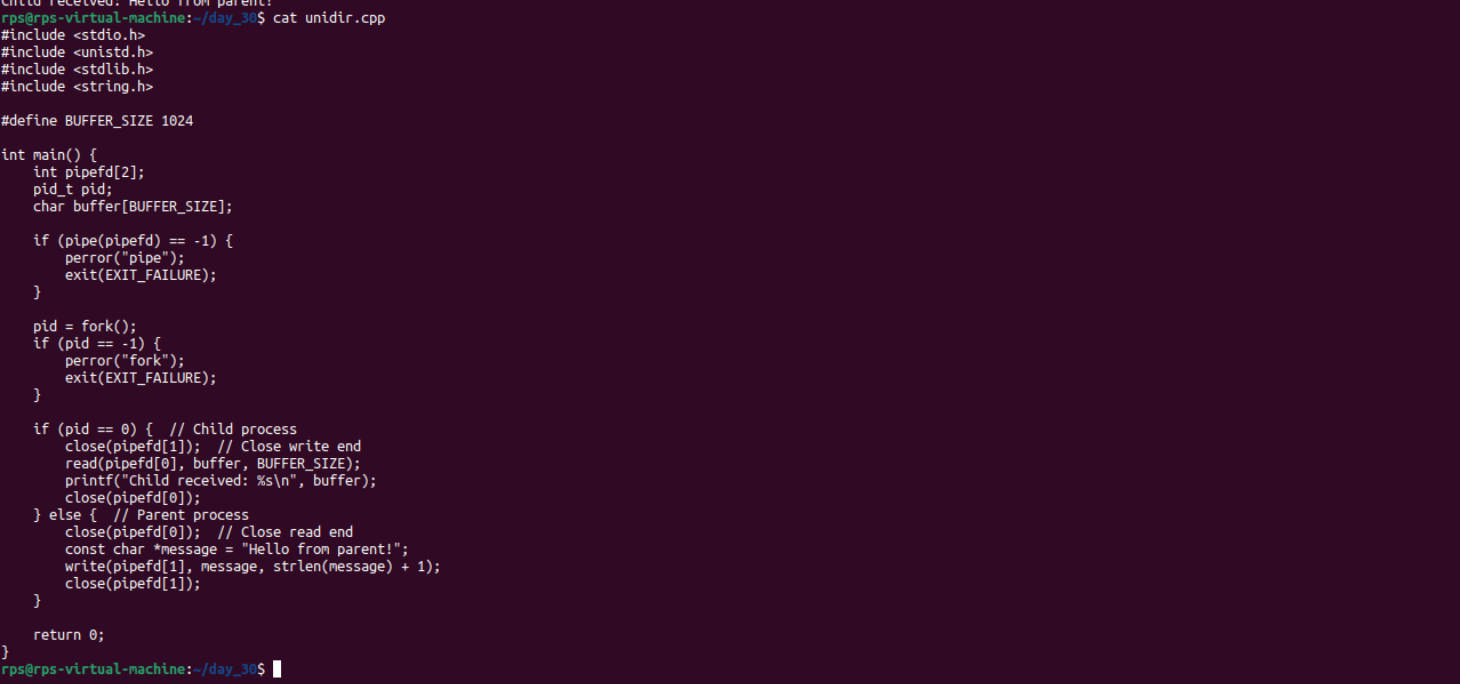
Message Queues: Create and utilize message queues for asynchronous communication and data exchange with message prioritization.

Synchronization: Implement appropriate synchronization mechanisms (e.g., semaphores, mutexes) to coordinate access to shared resources and prevent race conditions.

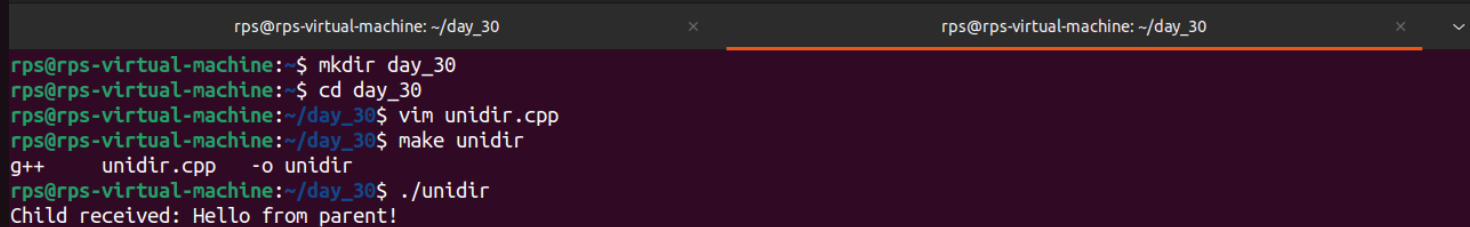
Error Handling: Incorporate robust error handling to manage potential IPC failures and resource leaks.

1.unidirectional:

Code:

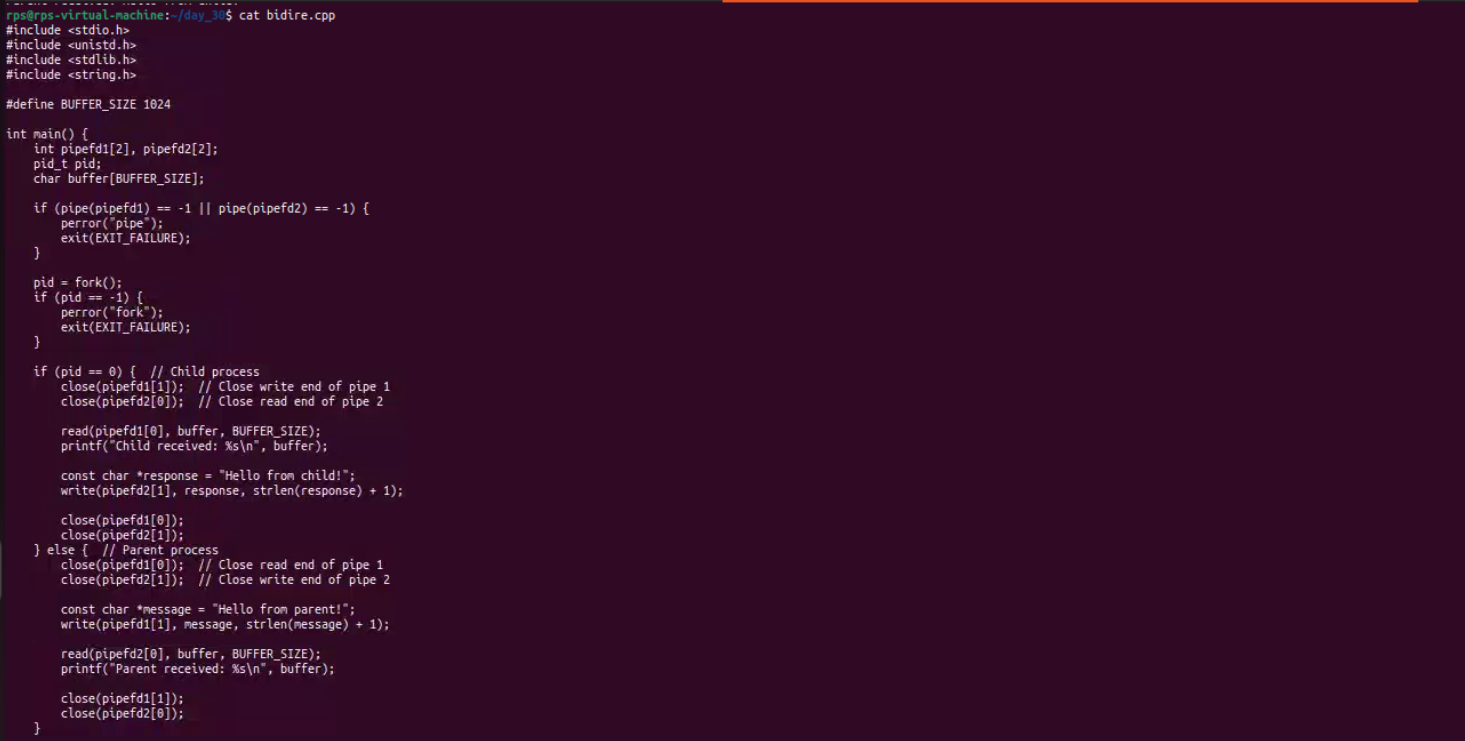


Output:



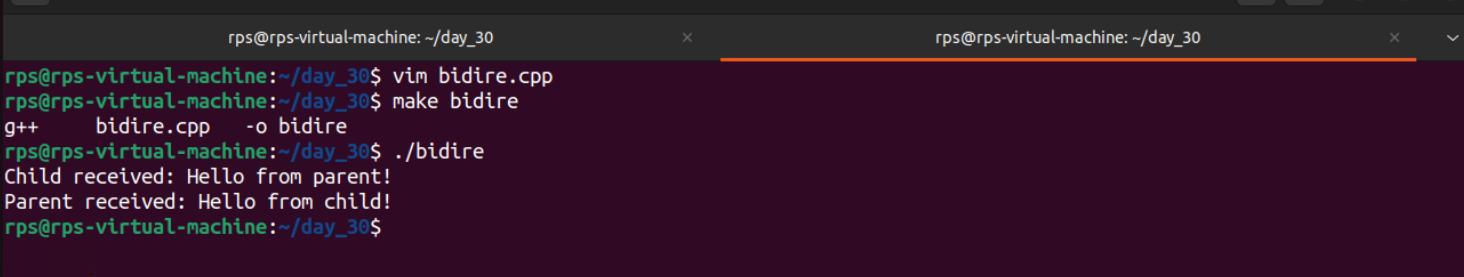
2.Bidirectional:

Code:



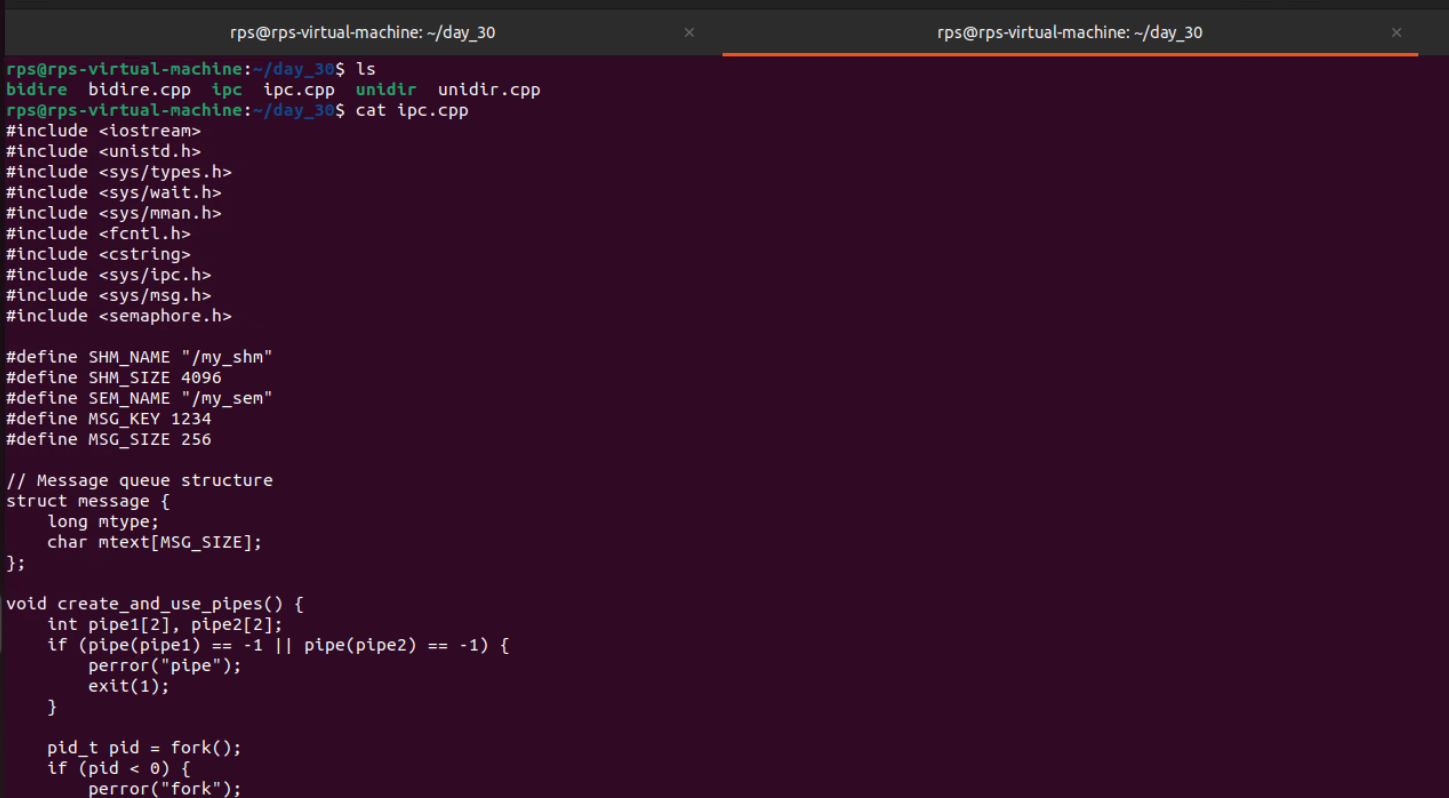


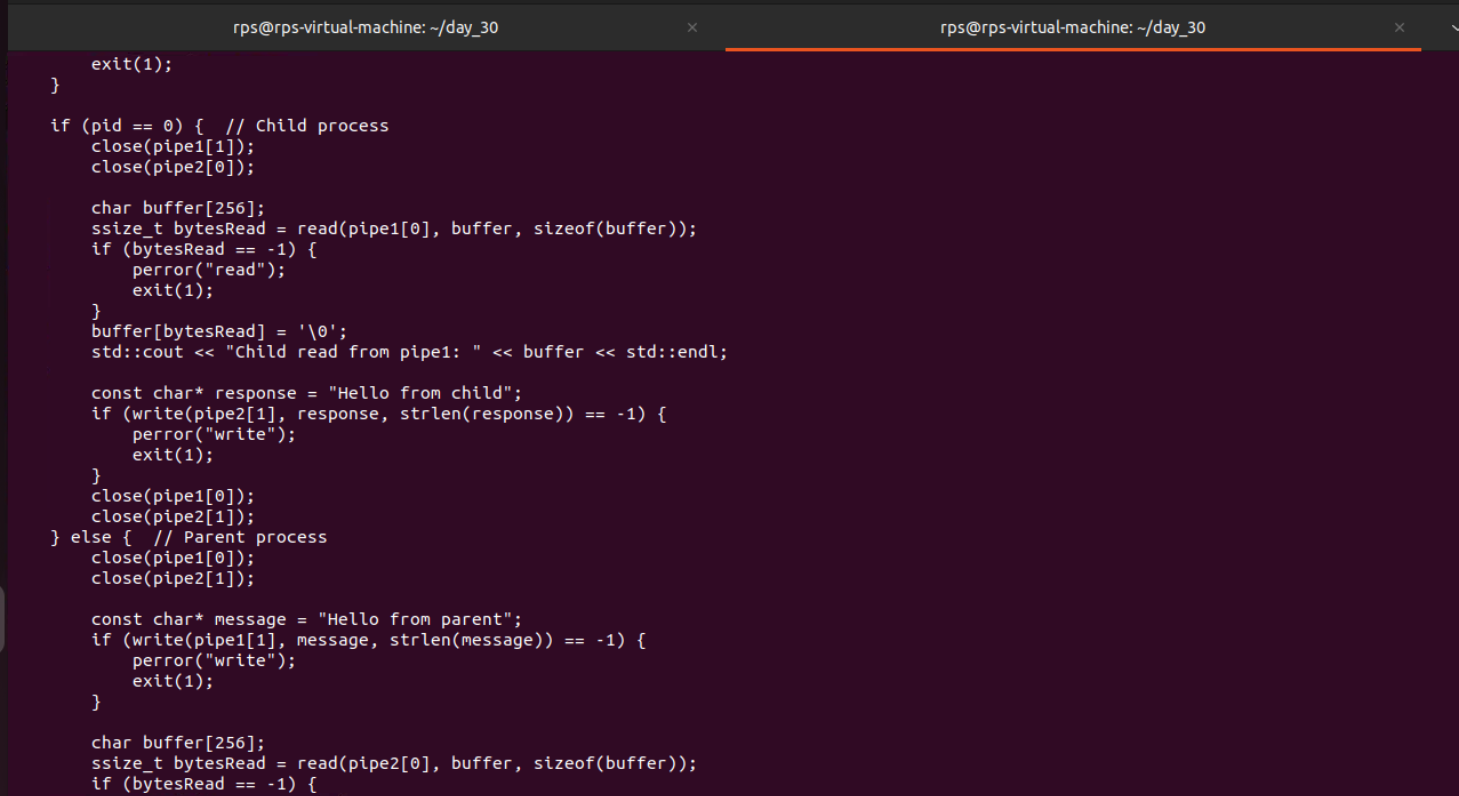
Output:

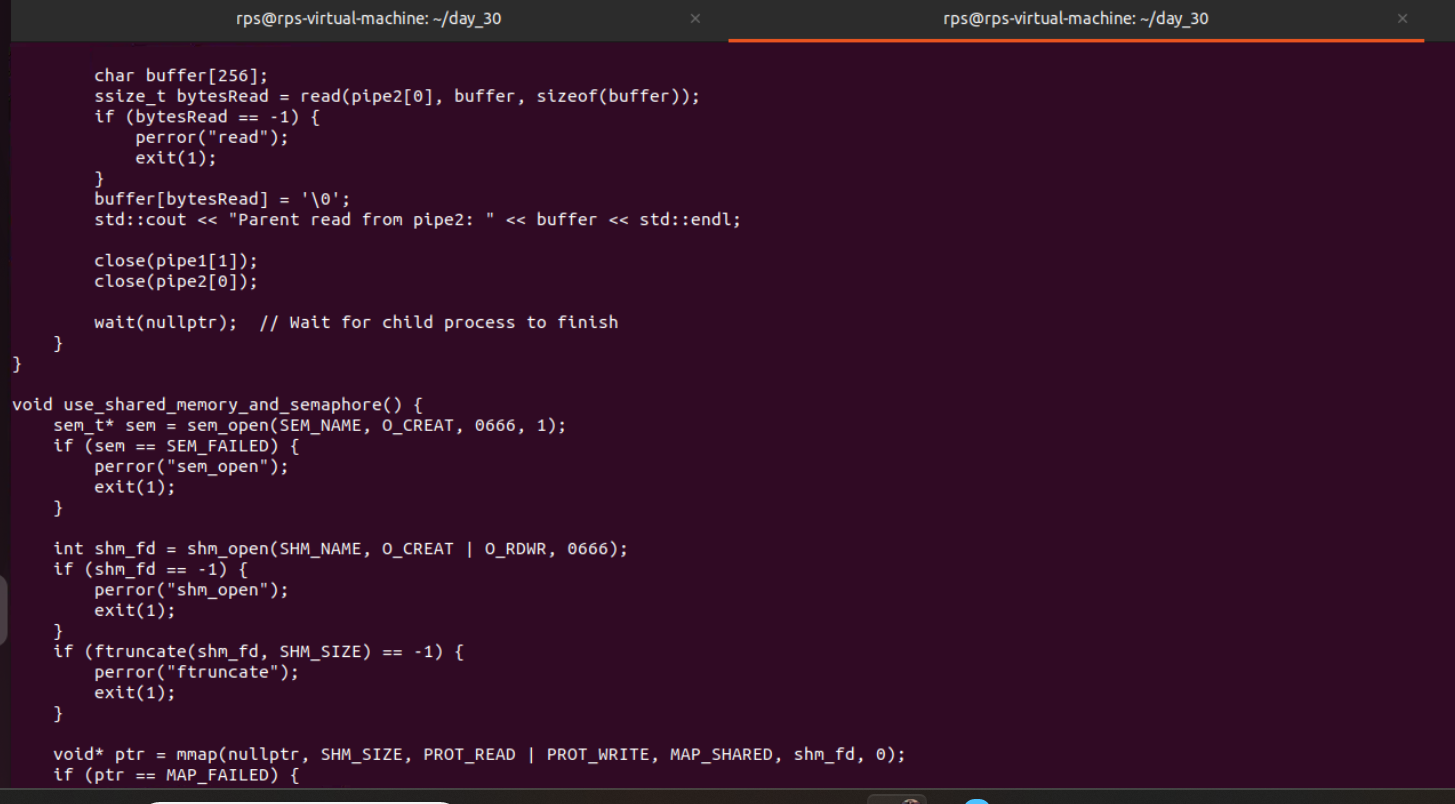


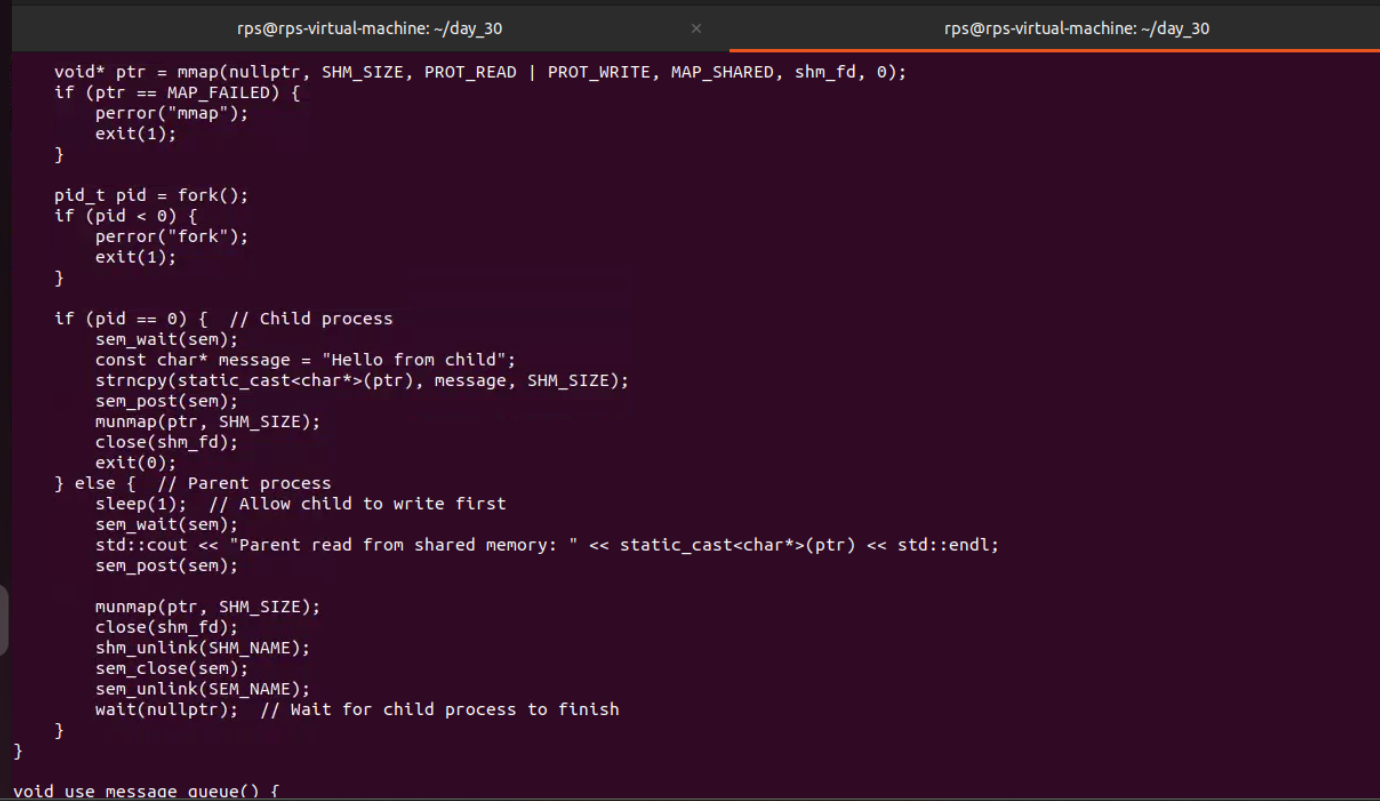
Single code IPC:

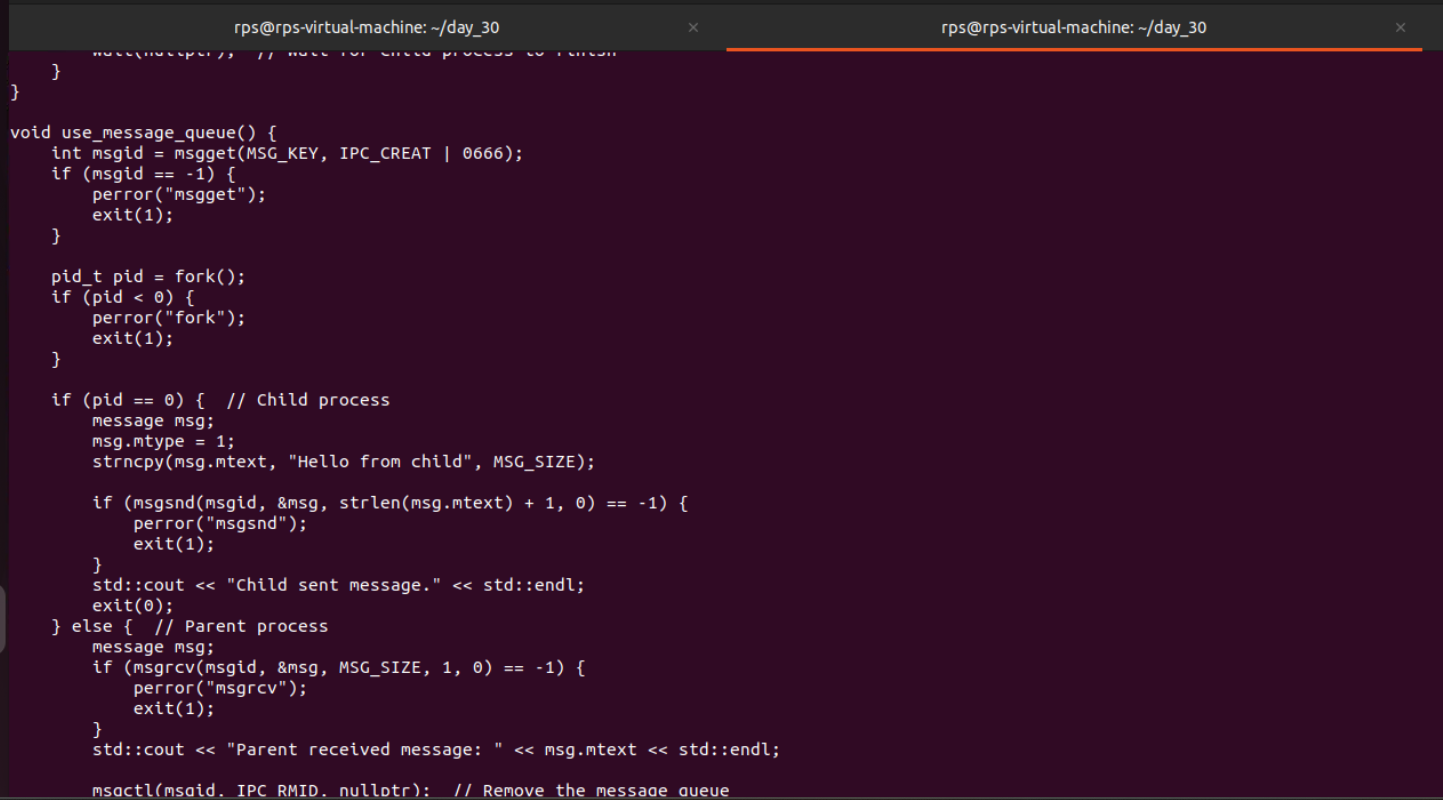
Code:

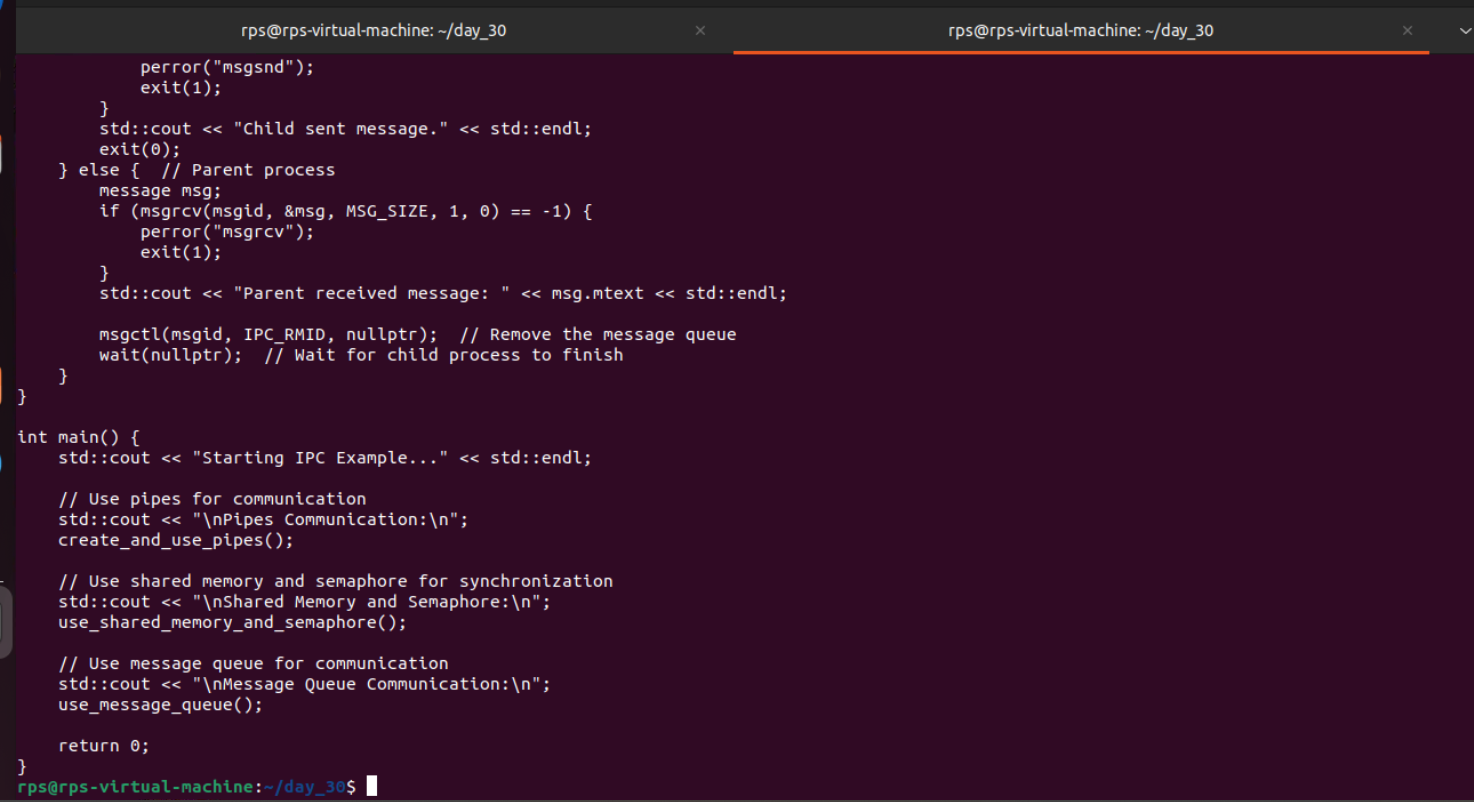


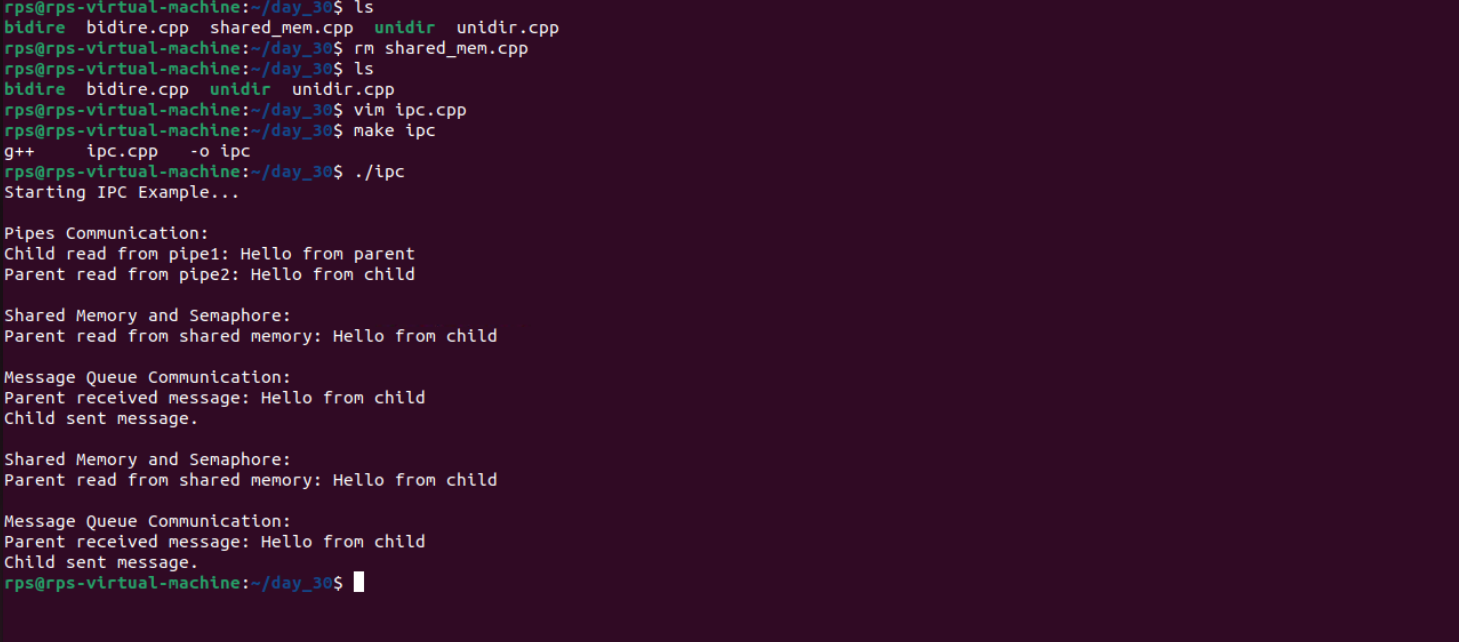










Output:

2.create a program that replicates the functionality of the standard cp command, but without using any standard library functions related to file I/O. Instead, you must employ system calls directly to perform file operations.

Requirements:

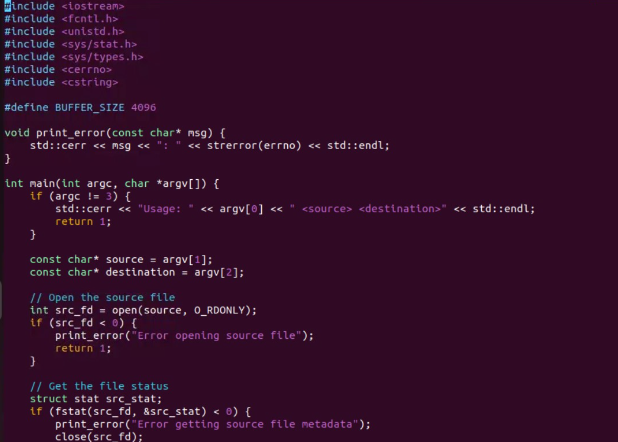
System calls: Utilize system calls like open, close, read, and write to interact with files.

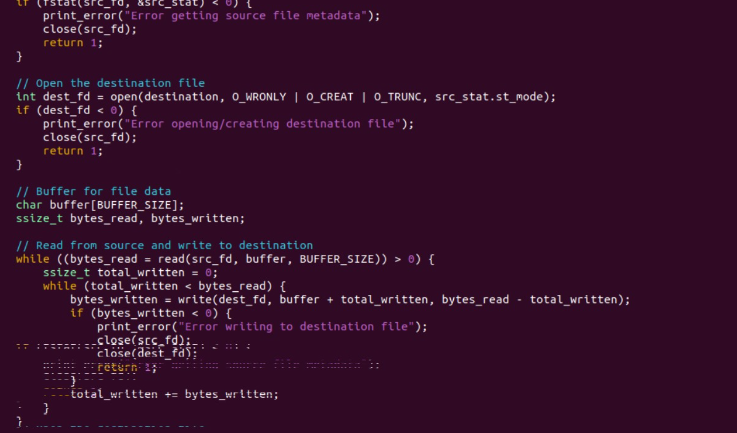
Error handling: Implement robust error handling for potential issues such as file not found, permission denied, disk full, etc.

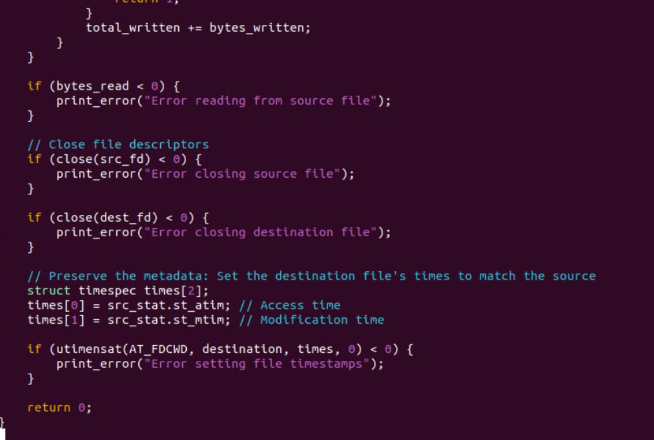
Efficiency: Optimize the copying process for performance, considering buffer sizes and read/write operations.

Metadata: Preserve file permissions, timestamps, and other relevant metadata during the copy process.

User interface: Provide a simple command-line interface with options for source and destination file paths.

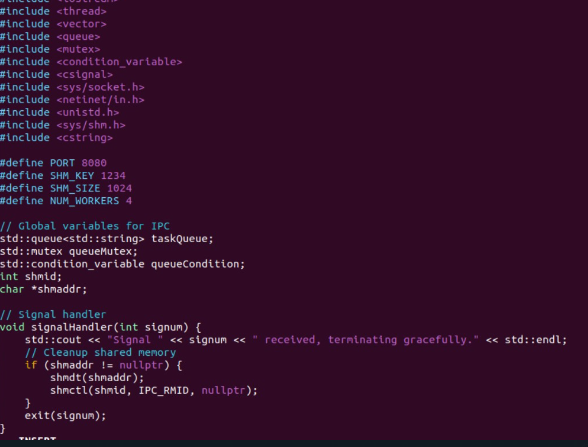
2. 



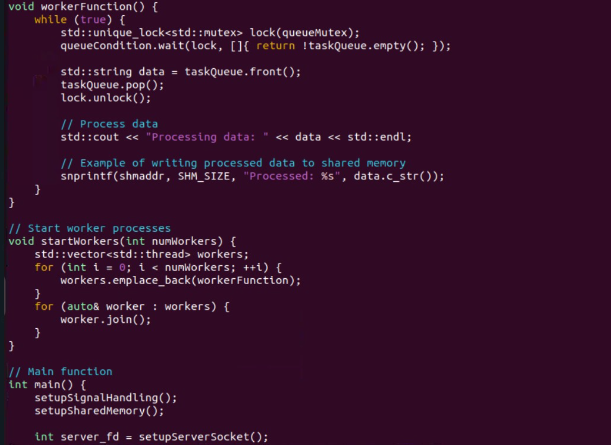


EXAM TASKS:

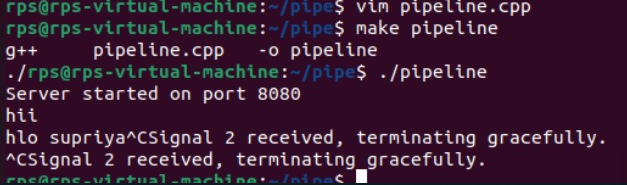
1.







Output:



2.

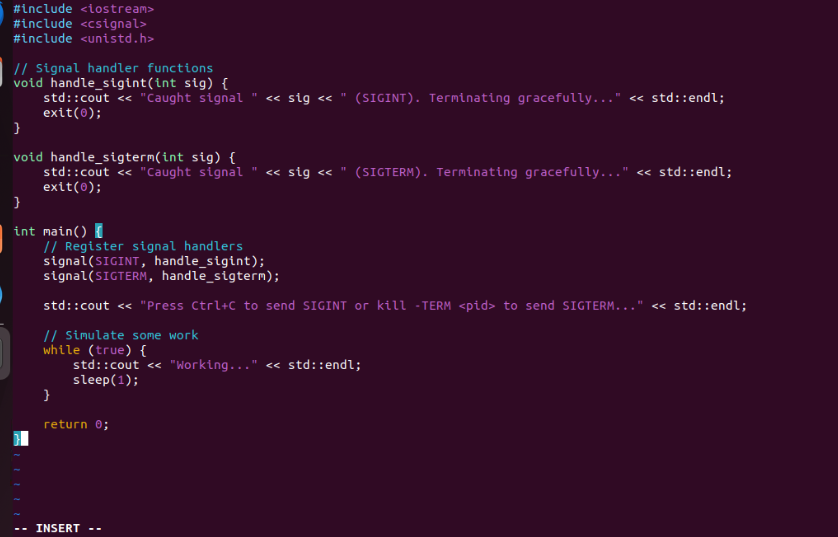
Coding Questions in C++

Signal Handling:

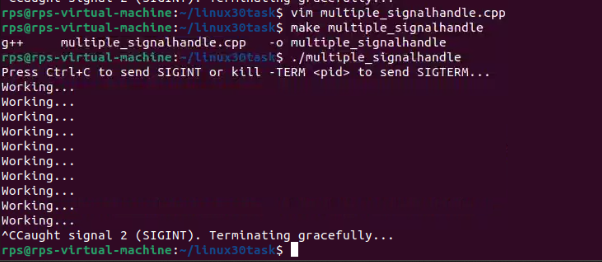
Write a C++ program that sets up a signal handler for SIGINT. The program should perform some tasks and print a message when SIGINT is caught, then terminate gracefully.



How would you modify your program to handle multiple different signals, each with a unique handling function?

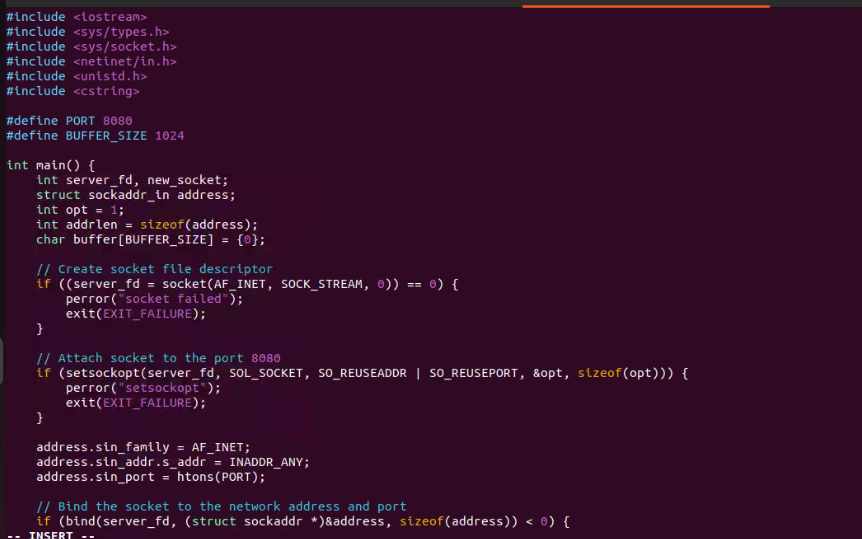


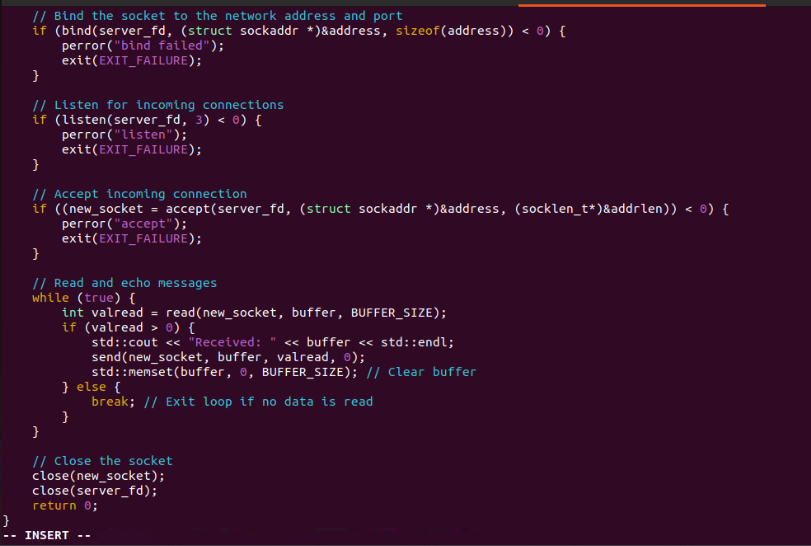
Output:



3.Sockets for Network Communication:

Implement a simple echo server in C++ that listens on a specific port, accepts client connections, and echoes back any messages received from clients.

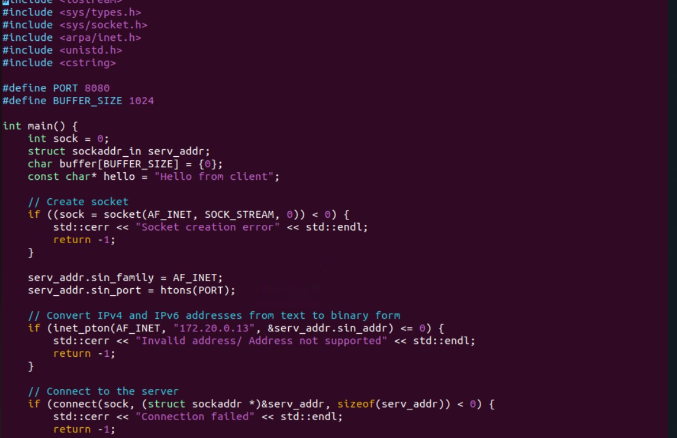


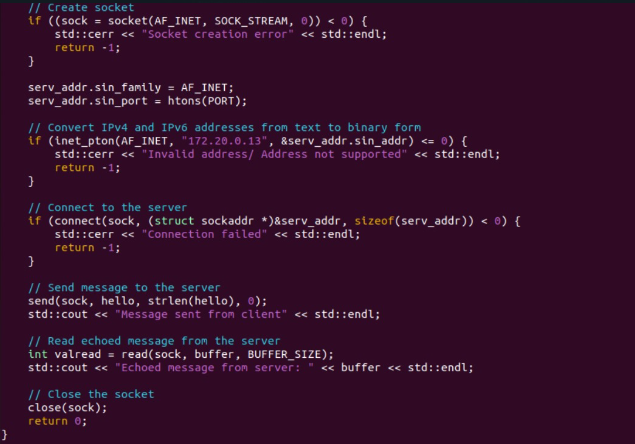


Output:

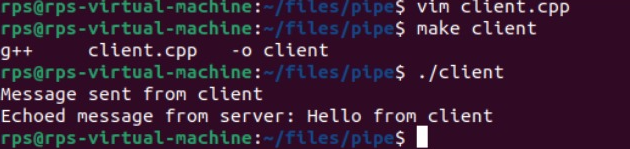


Write a client program that connects to the echo server, sends a message, and prints the echoed response.



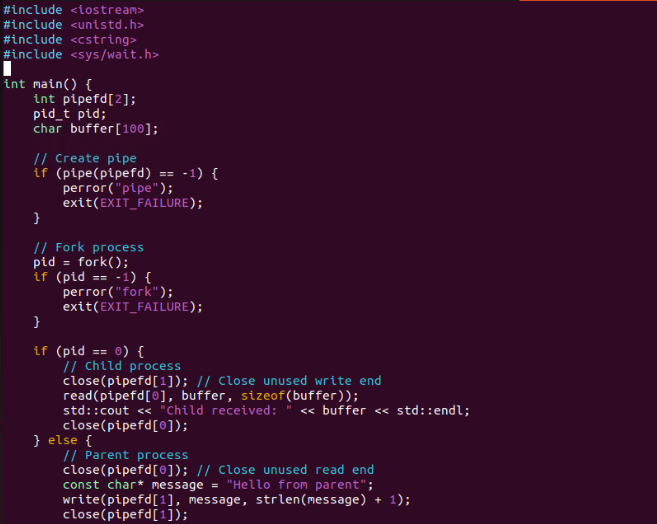


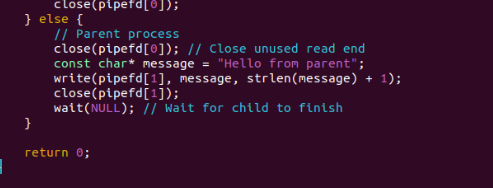
Output:



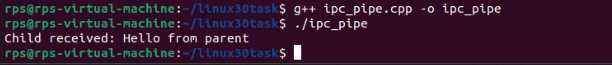
4. Inter-Process Communication (IPC):

Write a C++ program that creates a parent process and a child process. Use a pipe for IPC to send a message from the parent to the child, and have the child process print the message.

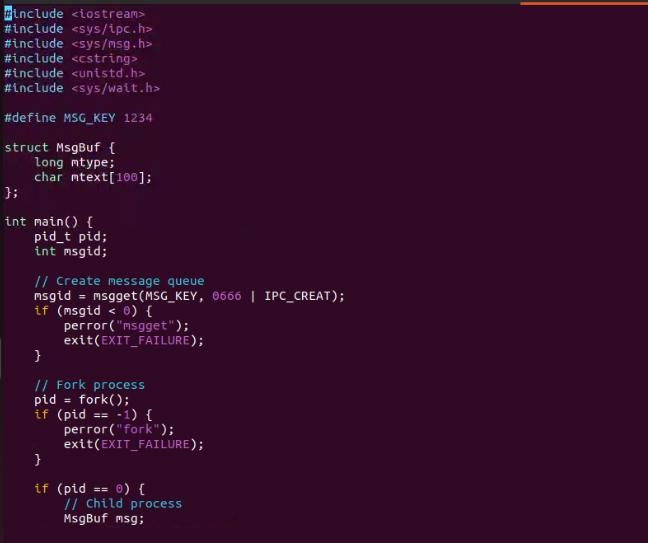




Output:



How would you modify the program to use a message queue instead of a pipe for communication between the parent and child processes?





Output:

