LINUX ASSIGNMENT:

Design and implement a robust, distributed system using C++ that effectively leverages signals, sockets, and inter-process communication (IPC) to manage and coordinate multiple processes for a real-time data processing pipeline.

System Requirements

Data Ingestion: Continuously receive data from multiple sources (e.g., network sockets, files, sensors) and distribute it across multiple worker processes.

Data Processing: Distribute incoming data to multiple worker processes, each responsible for specific data transformations or calculations.

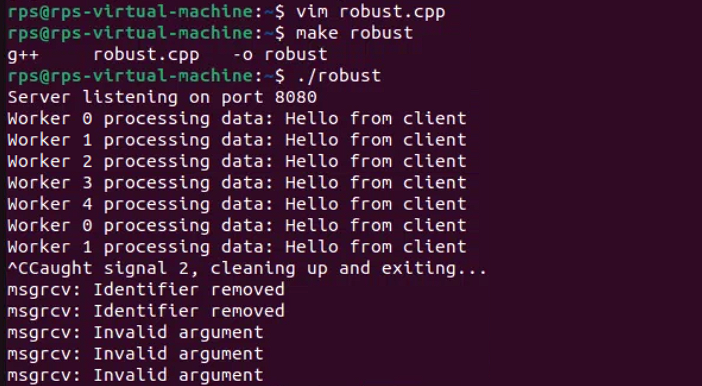
Error Handling: Implement robust error handling mechanisms using signals to gracefully handle unexpected events (e.g., process termination, network failures).

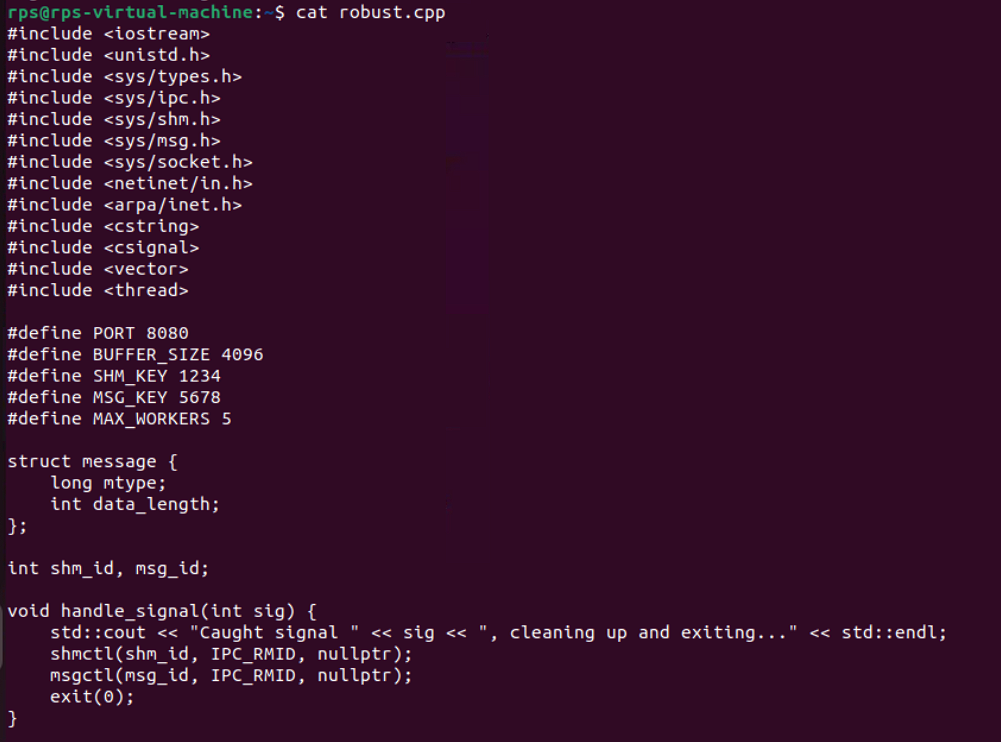
Inter-Process Communication: Utilize IPC (e.g., shared memory, message queues) for efficient communication and synchronization between processes.

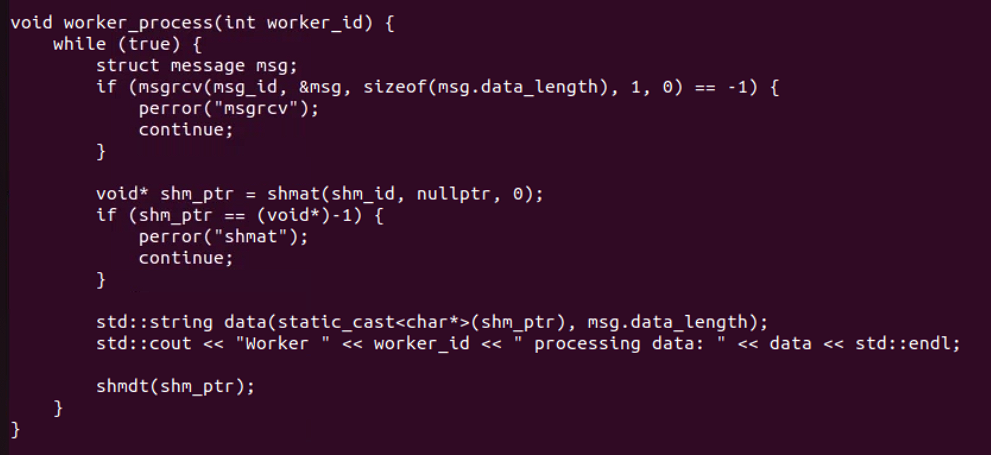
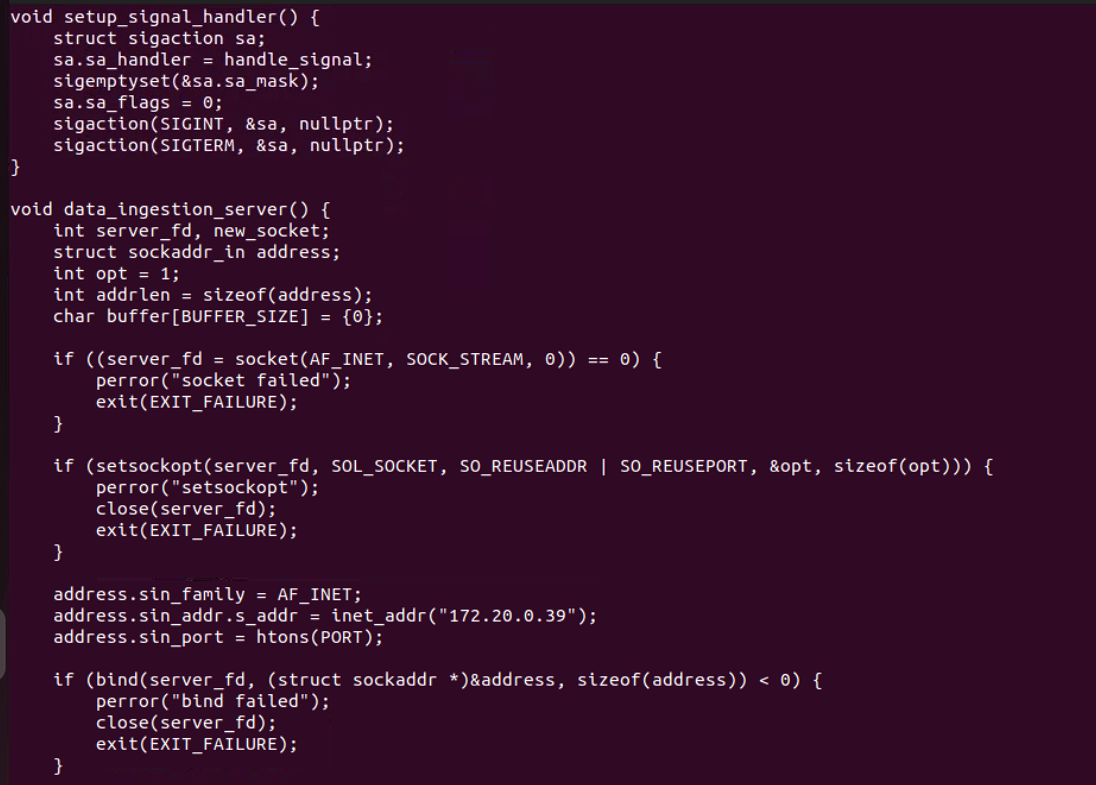
Performance Optimization: Optimize the system for low latency and high throughput, considering factors like network congestion, process scheduling, and data transfer efficiency.

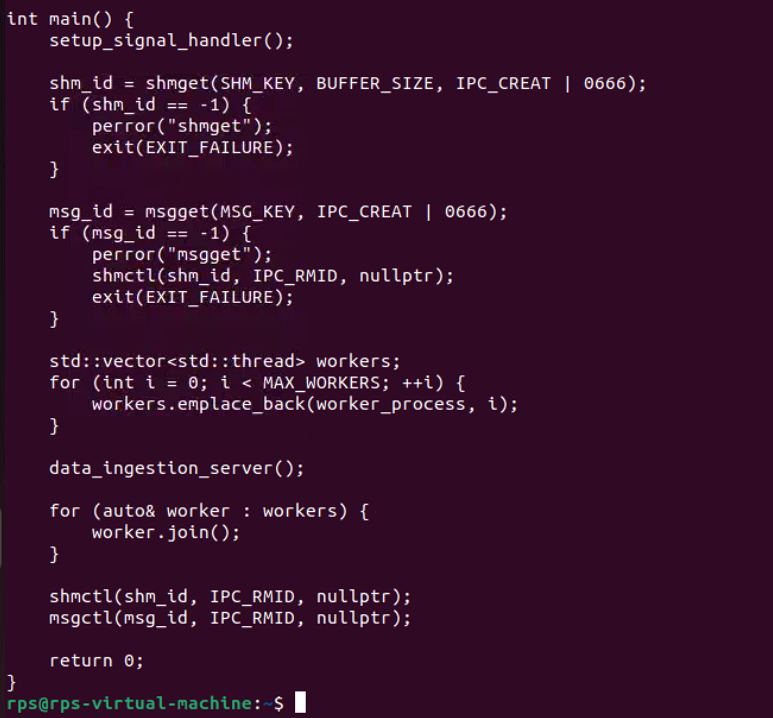
Scalability: Design the system to handle increasing data volumes and processing load by dynamically adjusting the number of worker processes**.**

**Server:**

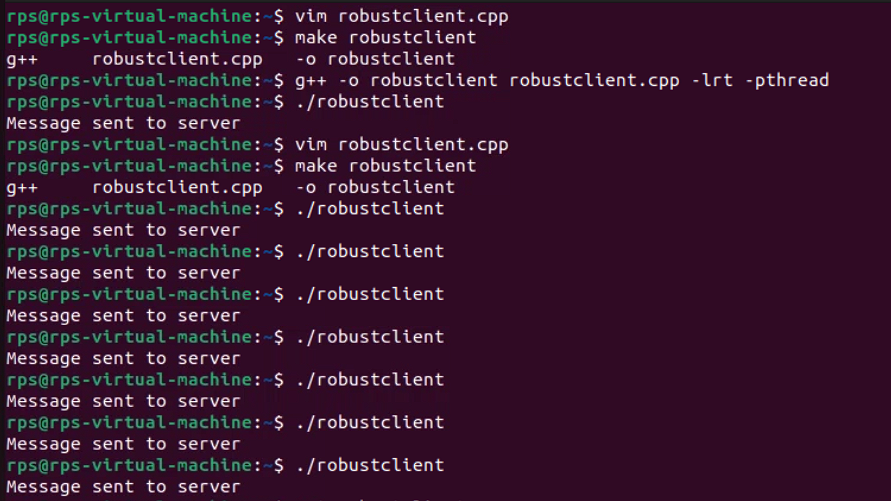
****

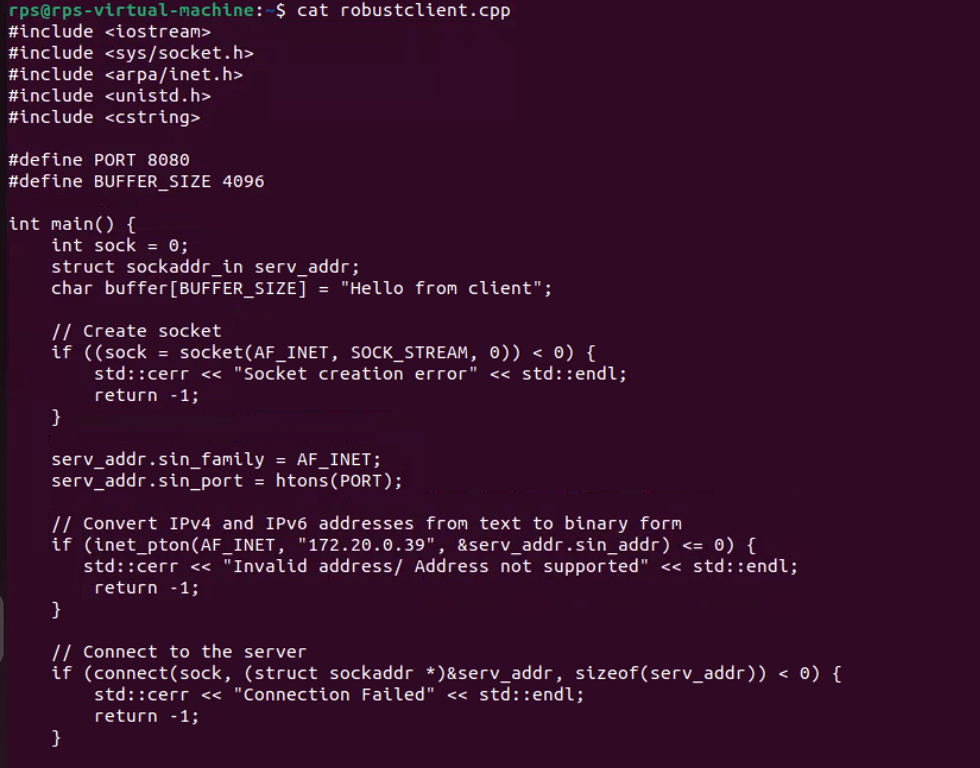


****

****

**Client:**

****

****

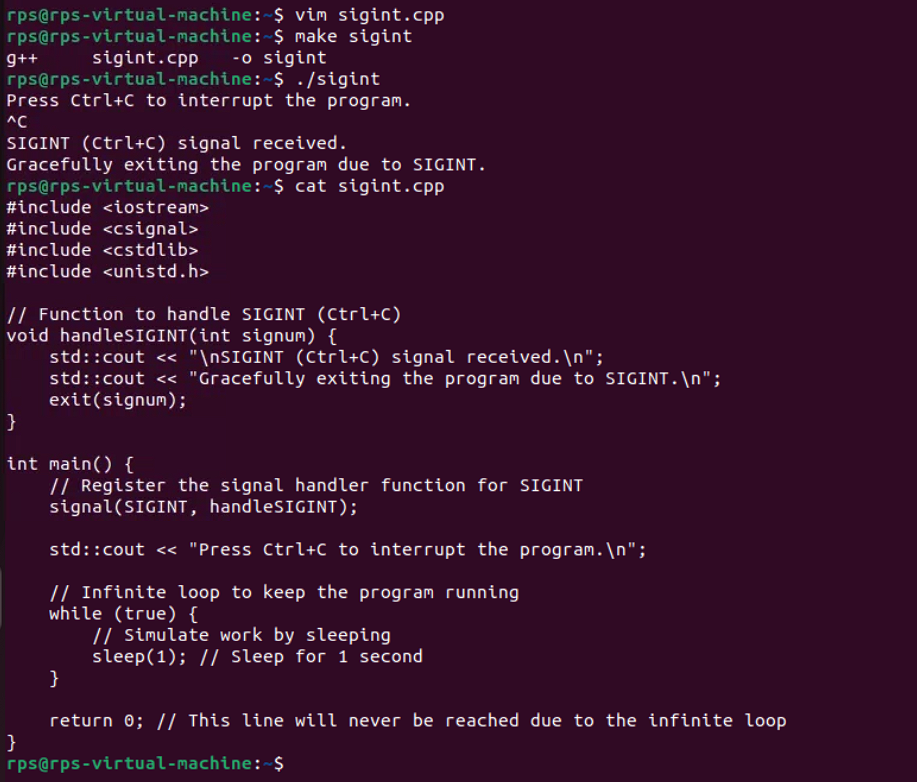
****

Coding Questions in C++

Signal Handling:

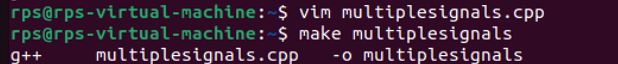
Write a C++ program that sets up a signal handler for SIGINT. The program should perform some tasks and print a message when SIGINT is caught, then terminate gracefully.

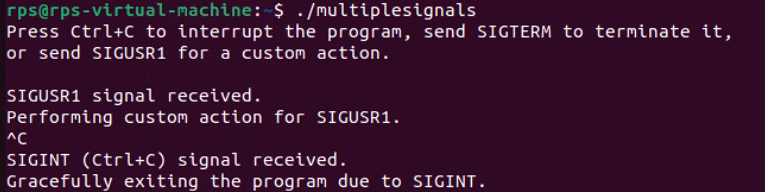
SIGIT

****

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

**Multiple Signals:**

****

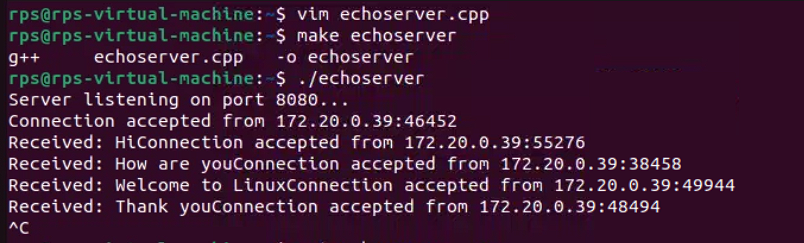
****

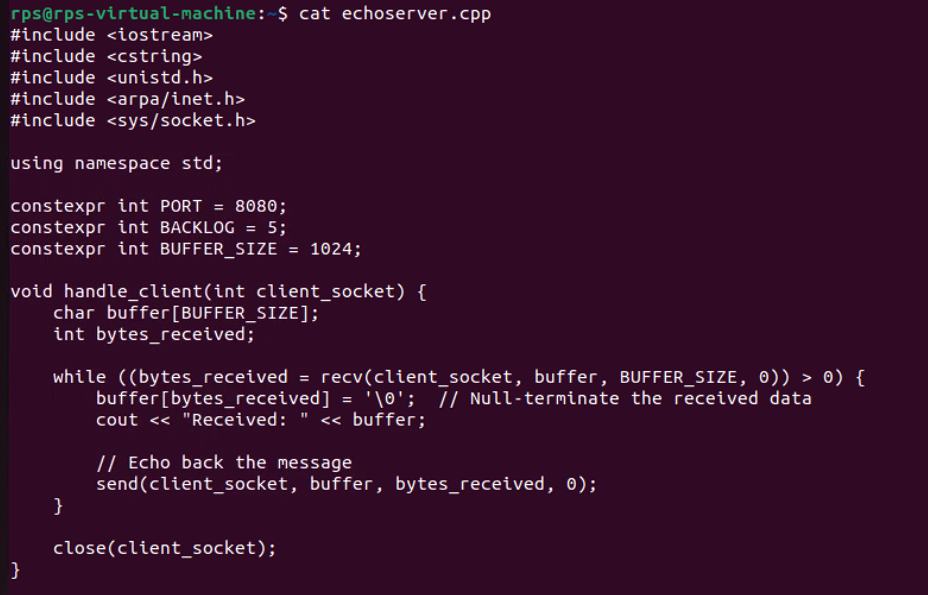
Sockets for Network Communication:

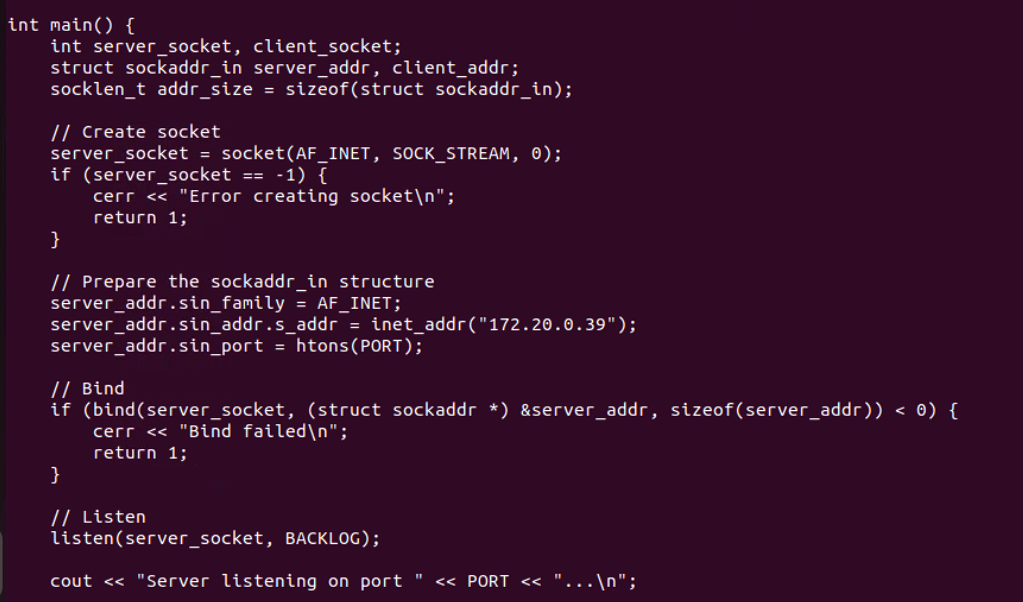
Implement a simple echo server in C++ that listens on a specific port, accepts client connections, and echoes back any messages received from clients.

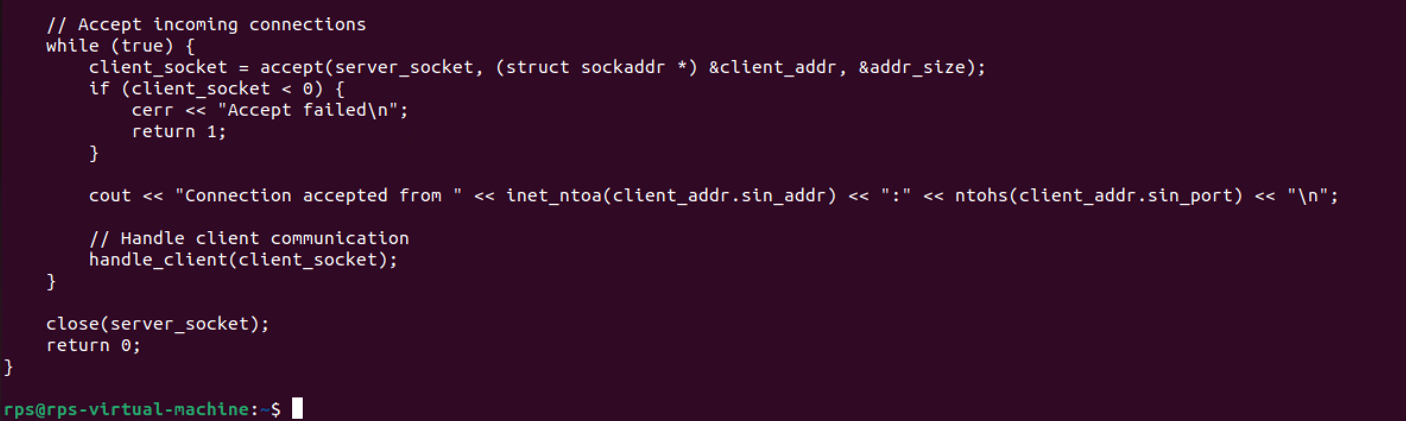
Write a client program that connects to the echo server, sends a message, and prints the echoed response.

Server:

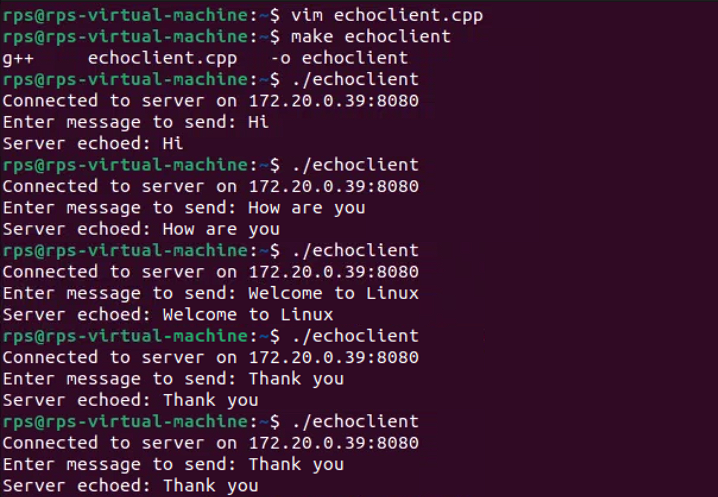
****

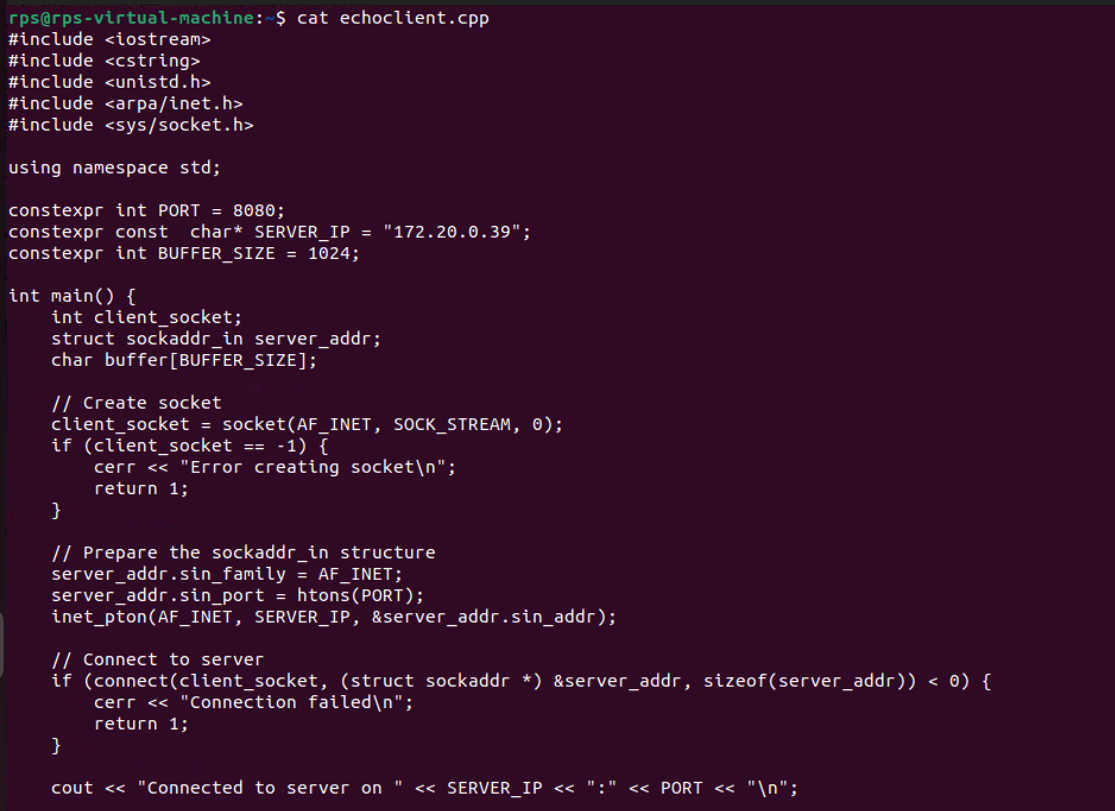
****

****

****

**Client:**

****

****

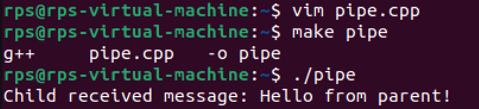
****

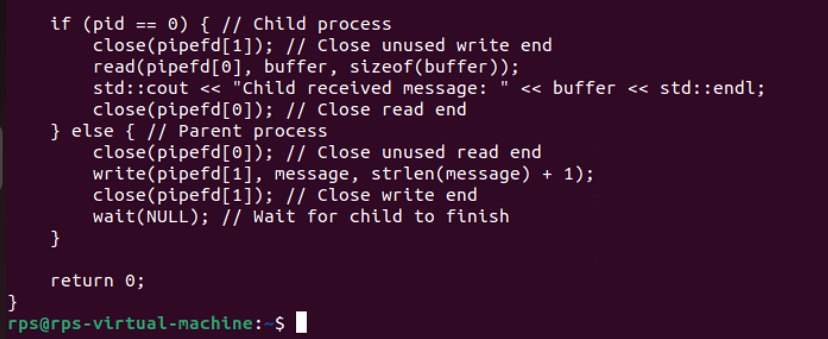
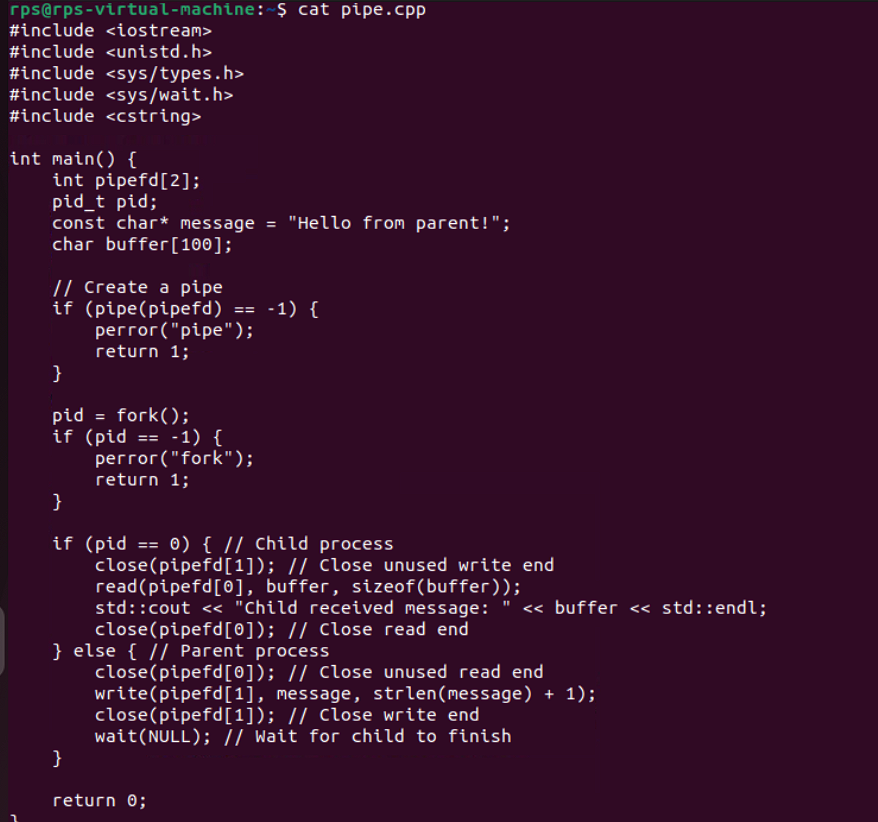
Inter-Process Communication (IPC):

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.

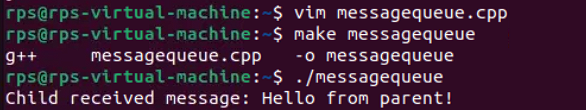
How would you modify the program to use a message queue instead of a pipe for communication between the parent and child processes?

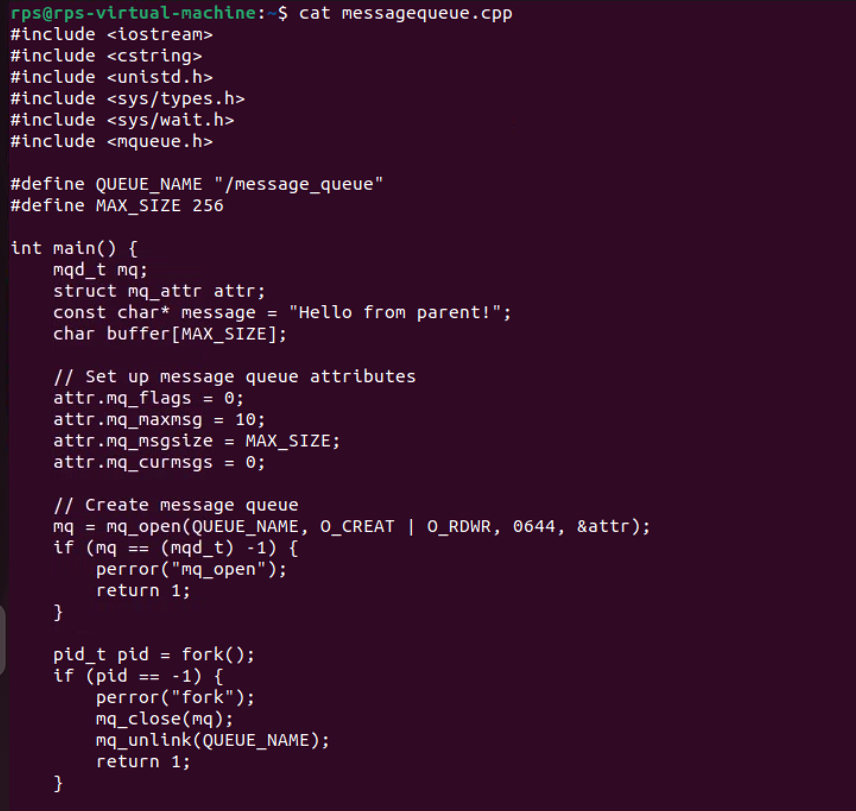
Using Pipe:

****

****

**Using messagequeue:**

****

****

****