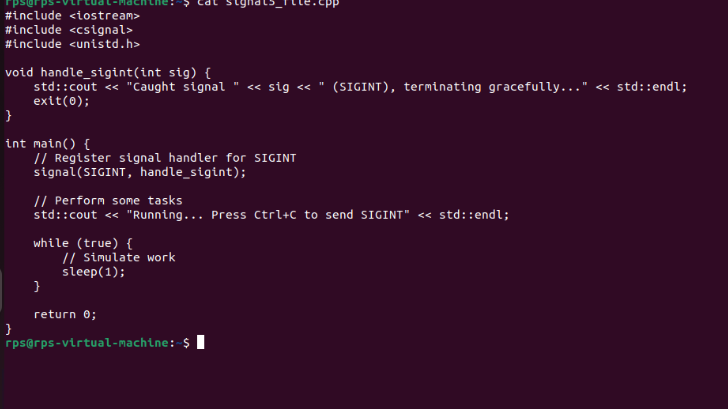
Coding Questions in C++

Signal Handling:

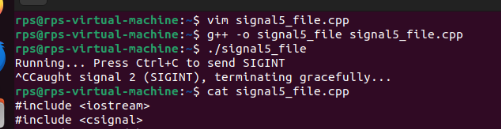
Write a C++ program that sets up a signal handler for SIGINT. The program should perform some tasks and print a message when SIGINT is caught, then terminate gracefully.

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

Code:



Output:



Part2:

To handle multiple signals in a C++ program, you follow these steps:

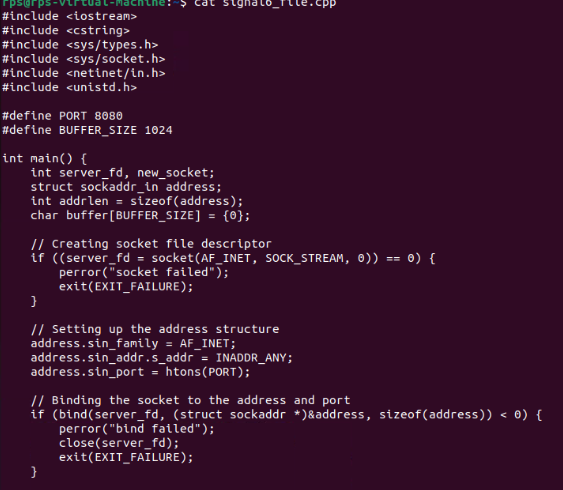
1. **Define Signal Handlers**: Write separate functions to handle each signal. Each function should perform the necessary tasks when a specific signal is caught.
2. **Main Program Logic**: Ensure your main program logic is running, usually in an infinite loop or while performing tasks, so it can catch the signals asynchronously.
3. **Register Signal Handlers**: Use the signal function to associate each signal with its corresponding handler function. For example, signal(SIGINT, handle\_sigint) registers the handle\_sigint function to handle the SIGINT signal.
4. **Compile and Run**: Compile the program with a C++ compiler (e.g., g++) and run the executable. Use appropriate commands or key combinations to send signals to the program (e.g., Ctrl+C for SIGINT, kill -TERM <pid> for SIGTERM).

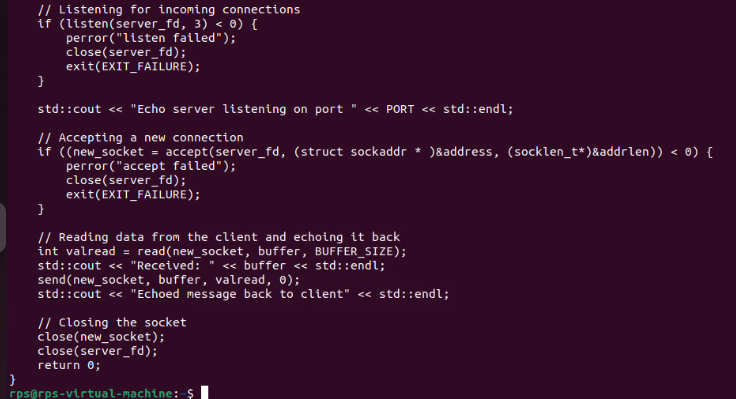
Sockets for Network Communication:

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

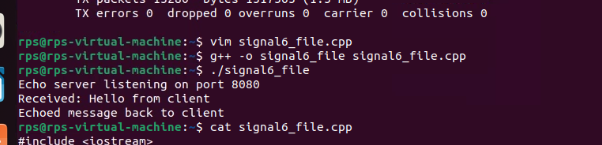
Write a client program that connects to the echo server, sends a message, and prints the echoed response.

Server code:

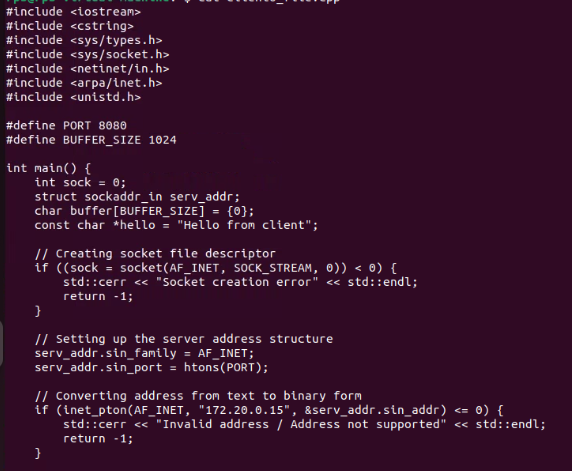


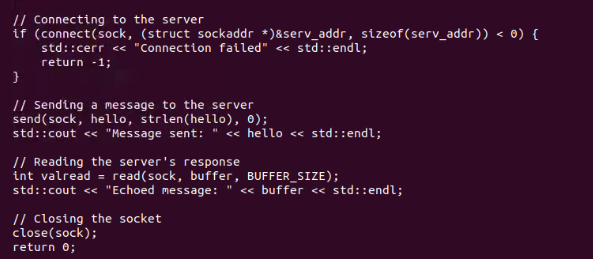


Output:

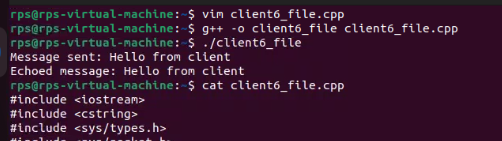


Client part:





Output:



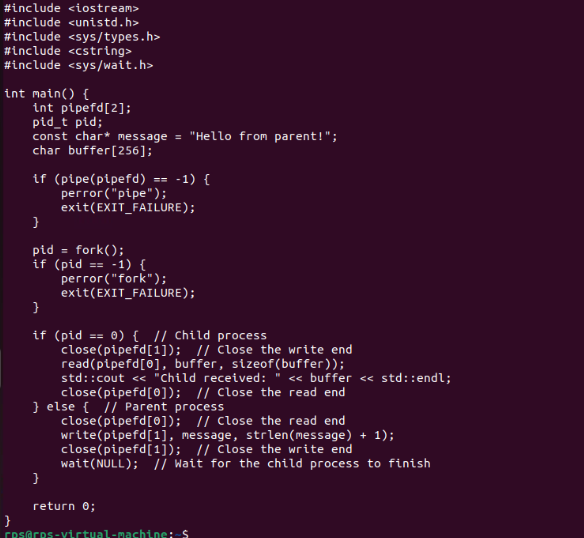
Inter-Process Communication (IPC):

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.

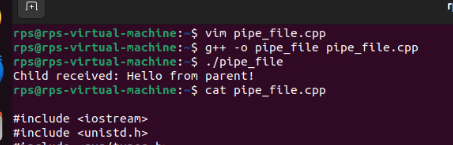
How would you modify the program to use a message queue instead of a pipe for communication between the parent and child processes.

Code:

Using pipe:

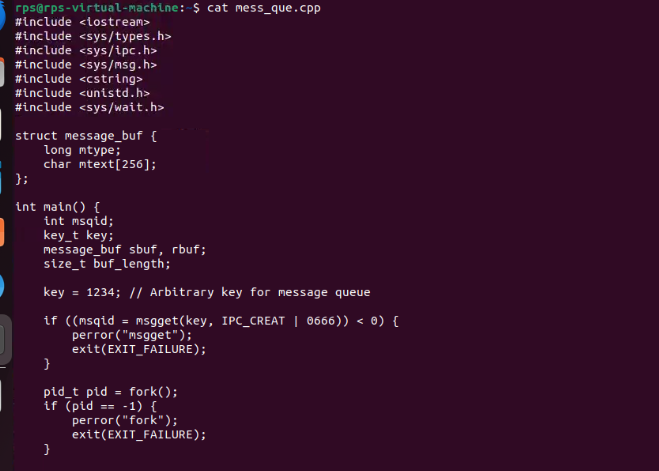


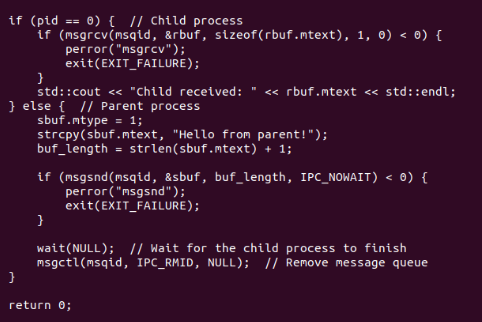
Output:



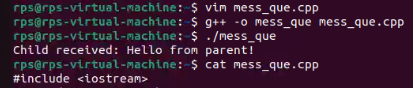
Code:

Using a message queue:





Output:



Part2:

Using a Pipe:

1. Pipes provide a unidirectional communication channel.
2. Pipes are suitable for simple parent-child communication but are limited in terms of flexibility and features.

Using a Message Queue:

1. Message queues allow communication between processes by sending and receiving messages.
2. Message queues are more flexible and robust compared to pipes, making them suitable for more complex communication needs.

In both cases, the parent process sends a message to the child process, which then reads and prints it. The primary difference lies in the mechanism used for communication: pipes for simpler, stream-based communication and message queues for more structured, message-based communication.

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.

