**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.**

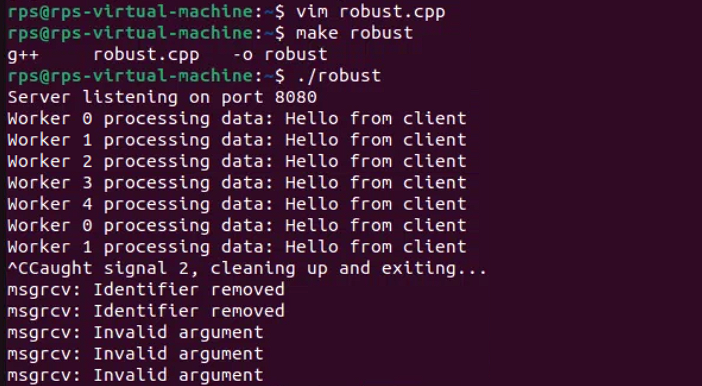
**Error Handling: Implement robust error handling mechanisms using signals to gracefully handle unexpected events (e.g., process termination, network failures).**

**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:**

****

#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/msg.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <cstring>

#include <csignal>

#include <vector>

#include <thread>

#define PORT 8080

#define BUFFER\_SIZE 4096

#define SHM\_KEY 1234

#define MSG\_KEY 5678

#define MAX\_WORKERS 5

struct message {

long mtype;

int data\_length;

};

int shm\_id, msg\_id;

void handle\_signal(int sig) {

std::cout << "Caught signal " << sig << ", cleaning up and exiting..." << std::endl;

shmctl(shm\_id, IPC\_RMID, nullptr);

msgctl(msg\_id, IPC\_RMID, nullptr);

exit(0);

}

void setup\_signal\_handler() {

struct sigaction sa;

sa.sa\_handler = handle\_signal;

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = 0;

sigaction(SIGINT, &sa, nullptr);

sigaction(SIGTERM, &sa, nullptr);

}

void data\_ingestion\_server() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt))) {

perror("setsockopt");

close(server\_fd);

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

while ((new\_socket = accept(server\_fd, (struct sockaddr )&address, (socklen\_t)&addrlen)) >= 0) {

ssize\_t bytes\_read = read(new\_socket, buffer, BUFFER\_SIZE);

if (bytes\_read > 0) {

void\* shm\_ptr = shmat(shm\_id, nullptr, 0);

if (shm\_ptr == (void\*)-1) {

perror("shmat");

close(new\_socket);

continue;

}

memcpy(shm\_ptr, buffer, bytes\_read);

struct message msg;

msg.mtype = 1;

msg.data\_length = bytes\_read;

if (msgsnd(msg\_id, &msg, sizeof(msg.data\_length), 0) == -1) {

perror("msgsnd");

}

shmdt(shm\_ptr);

}

close(new\_socket);

}

if (new\_socket < 0) {

perror("accept");

}

}

void worker\_process(int worker\_id) {

while (true) {

struct message msg;

if (msgrcv(msg\_id, &msg, sizeof(msg.data\_length), 1, 0) == -1) {

perror("msgrcv");

continue;

}

void\* shm\_ptr = shmat(shm\_id, nullptr, 0);

if (shm\_ptr == (void\*)-1) {

perror("shmat");

continue;

}

std::string data(static\_cast<char\*>(shm\_ptr), msg.data\_length);

std::cout << "Worker " << worker\_id << " processing data: " << data << std::endl;

shmdt(shm\_ptr);

}

}

int main() {

setup\_signal\_handler();

shm\_id = shmget(SHM\_KEY, BUFFER\_SIZE, IPC\_CREAT | 0666);

if (shm\_id == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

msg\_id = msgget(MSG\_KEY, IPC\_CREAT | 0666);

if (msg\_id == -1) {

perror("msgget");

shmctl(shm\_id, IPC\_RMID, nullptr);

exit(EXIT\_FAILURE);

}

std::vector<std::thread> workers;

for (int i = 0; i < MAX\_WORKERS; ++i) {

workers.emplace\_back(worker\_process, i);

}

data\_ingestion\_server();

for (auto& worker : workers) {

worker.join();

}

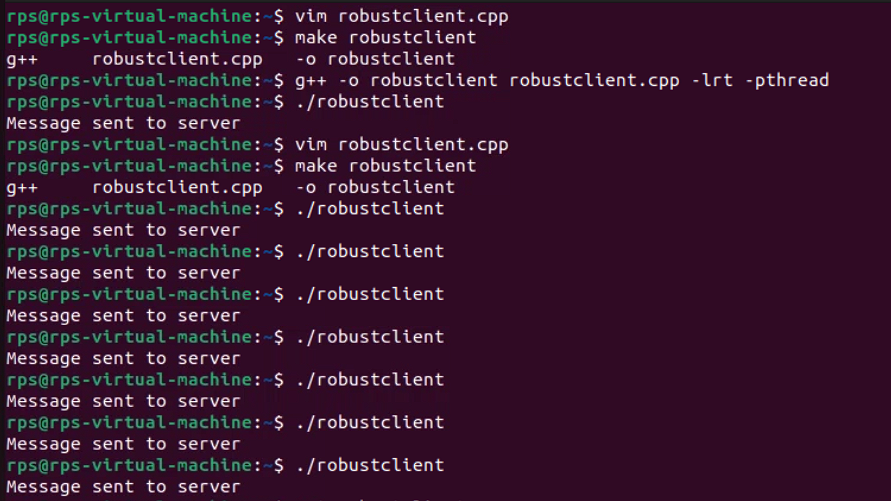
shmctl(shm\_id, IPC\_RMID, nullptr);

msgctl(msg\_id, IPC\_RMID, nullptr);

return 0;

}

**Client:**

****

#include <iostream>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <cstring>

#define PORT 8080

#define BUFFER\_SIZE 4096

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = "Hello from client";

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

std::cerr << "Socket creation error" << std::endl;

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

std::cerr << "Invalid address/ Address not supported" << std::endl;

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

std::cerr << "Connection Failed" << std::endl;

return -1;

}

// Send data to the server

send(sock, buffer, strlen(buffer), 0);

std::cout << "Message sent to server" << std::endl;

// Close the socket

close(sock);

return 0;

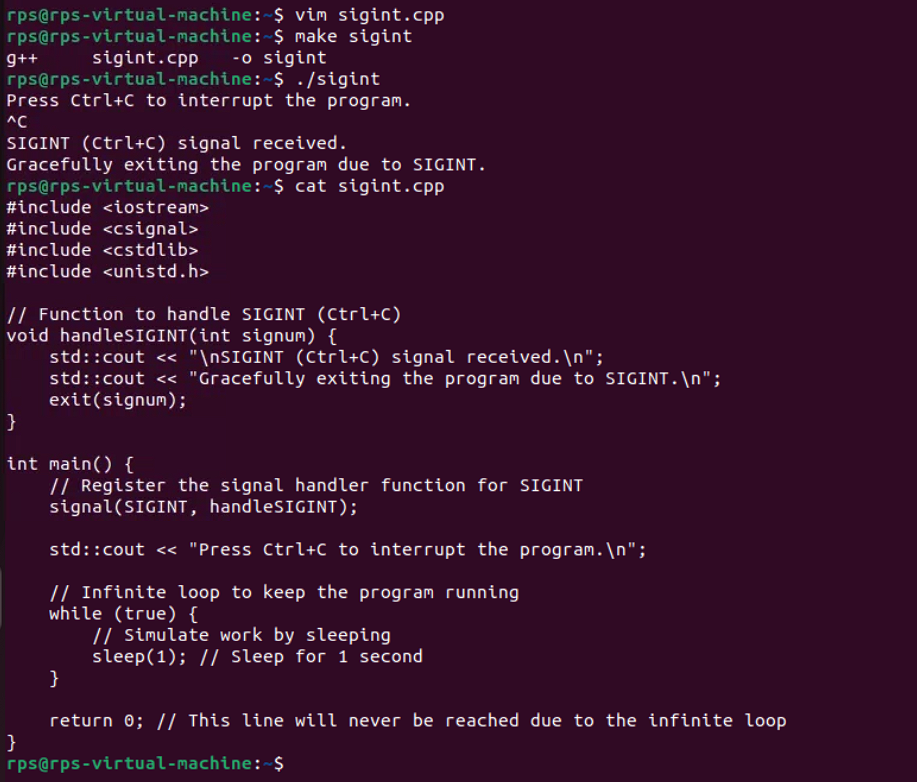
}

**Coding Questions in C++**

**Signal Handling:**

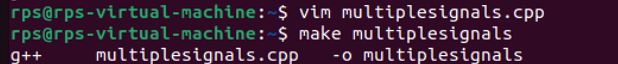
**Write a C++ program that sets up a signal handler for SIGINT. The program should perform some tasks and print a message when SIGINT is caught, then terminate gracefully.**

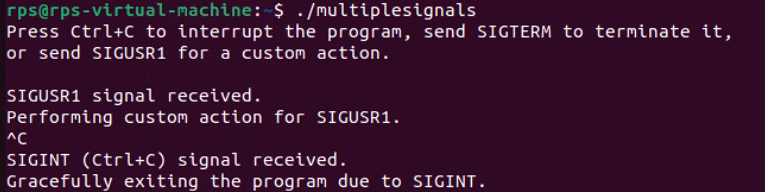
**SIGINT:**

****

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

**Multiple Signals:**

****

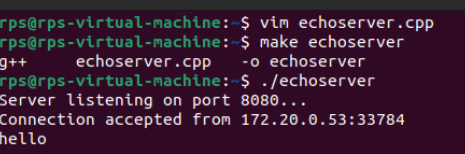
****

**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**



#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

using namespace std;

constexpr int PORT = 8080;

constexpr int BACKLOG = 5;

constexpr int BUFFER\_SIZE = 1024;

void handle\_client(int client\_socket) {

char buffer[BUFFER\_SIZE];

int bytes\_received;

while ((bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0)) > 0) {

buffer[bytes\_received] = '\0'; // Null-terminate the received data

cout << "Received: " << buffer;

// Echo back the message

send(client\_socket, buffer, bytes\_received, 0);

}

close(client\_socket);

}

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_size = sizeof(struct sockaddr\_in);

// Create socket

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

cerr << "Error creating socket\n";

return 1;

}

// Prepare the sockaddr\_in structure

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind

if (bind(server\_socket, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)) < 0) {

cerr << "Bind failed\n";

return 1;

}

// Listen

listen(server\_socket, BACKLOG);

cout << "Server listening on port " << PORT << "...\n";

// Accept incoming connections

while (true) {

client\_socket = accept(server\_socket, (struct sockaddr \*) &client\_addr, &addr\_size);

if (client\_socket < 0) {

cerr << "Accept failed\n";

return 1;

}

cout << "Connection accepted from " << inet\_ntoa(client\_addr.sin\_addr) << ":" << ntohs(client\_addr.sin\_port) << "\n";

// Handle client communication

handle\_client(client\_socket);

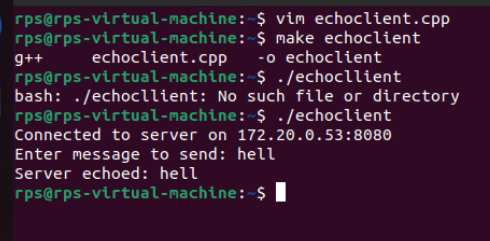
}

close(server\_socket);

return 0;

}

CLIENT:



#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

using namespace std;

constexpr int PORT = 8080;

constexpr const char\* SERVER\_IP = "127.0.0.1"; // Use const char\* instead of char\*

constexpr int BUFFER\_SIZE = 1024;

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

// Create socket

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket == -1) {

cerr << "Error creating socket\n";

return 1;

}

// Prepare the sockaddr\_in structure

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr);

// Connect to server

if (connect(client\_socket, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)) < 0) {

cerr << "Connection failed\n";

return 1;

}

cout << "Connected to server on " << SERVER\_IP << ":" << PORT << "\n";

// Send message to server

cout << "Enter message to send: ";

cin.getline(buffer, BUFFER\_SIZE);

send(client\_socket, buffer, strlen(buffer), 0);

// Receive echoed message from server

int bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received > 0) {

buffer[bytes\_received] = '\0'; // Null-terminate received data

cout << "Server echoed: " << buffer << "\n";

}

close(client\_socket);

return 0;

}