SIGPROCMASK

task is to implement a function to process a signal and mark the processed elements using a specific marker. The signal is represented as a vector of integers. You need to:

Define a marker value (SIGPROCMARK) to mark the processed signal elements.

Implement a function processSignal that processes each element of the signal by doubling its value and then marking it with SIGPROCMARK.

Implement a function displaySignal to print the signal values to the console.

Demonstrate the usage of these functions in a main function with an example signal.

Requirements:

The marker value should be defined as a constant.

The processSignal function should use bitwise operations to mark the processed elements.

The displaySignal function should print the signal values separated by spaces.

Input:

An example signal represented as a vector of integers, e.g., {1, 2, 3, 4, 5}.

#include <iostream>

#include <vector>

// Define a marker value for processed signals

const int SIGPROCMARK = 1;

// Function to process a signal and mark it

void processSignal(std::vector<int>& signal) {

for (size\_t i = 0; i < signal.size(); ++i) {

// Process the signal (example: double the value)

signal[i] \*= 2;

// Mark the processed signal

signal[i] |= SIGPROCMARK;

}

}

// Function to display the signal

void displaySignal(const std::vector<int>& signal) {

for (size\_t i = 0; i < signal.size(); ++i) {

std::cout << signal[i] << " ";

}

std::cout << std::endl;

}

int main() {

// Example signal

std::vector<int> signal = {1, 2, 3, 4, 5};

std::cout << "Original signal: ";

displaySignal(signal);

// Process the signal

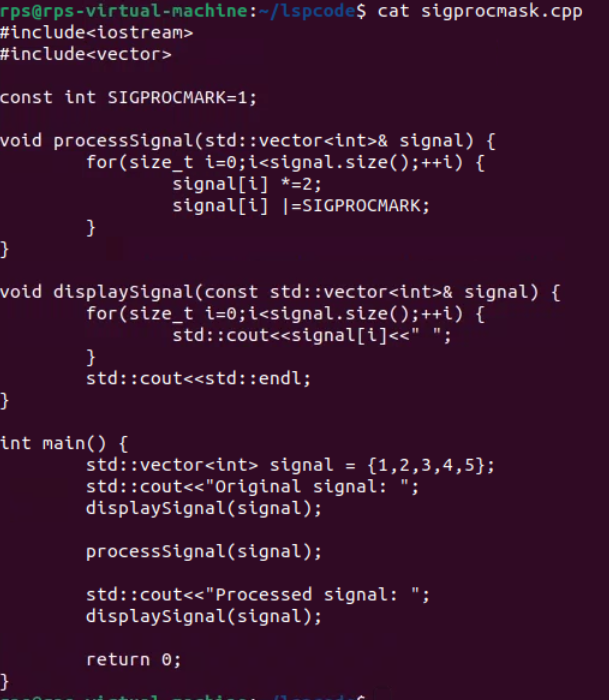
processSignal(signal);

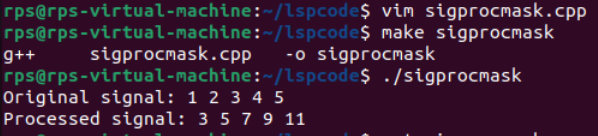
std::cout << "Processed signal: ";

displaySignal(signal);

return 0;

}





Signal Processing with Threshold Marking

You are tasked with extending the signal processing project to include a threshold marking mechanism. Your goal is to:

Define a marker value (SIGPROCMARK) to mark the processed signal elements.

Implement a function processSignalWithThreshold that processes each element of the signal by doubling its value only if it is greater than a given threshold, and then marking it with SIGPROCMARK.

Implement a function displaySignal to print the signal values to the console.

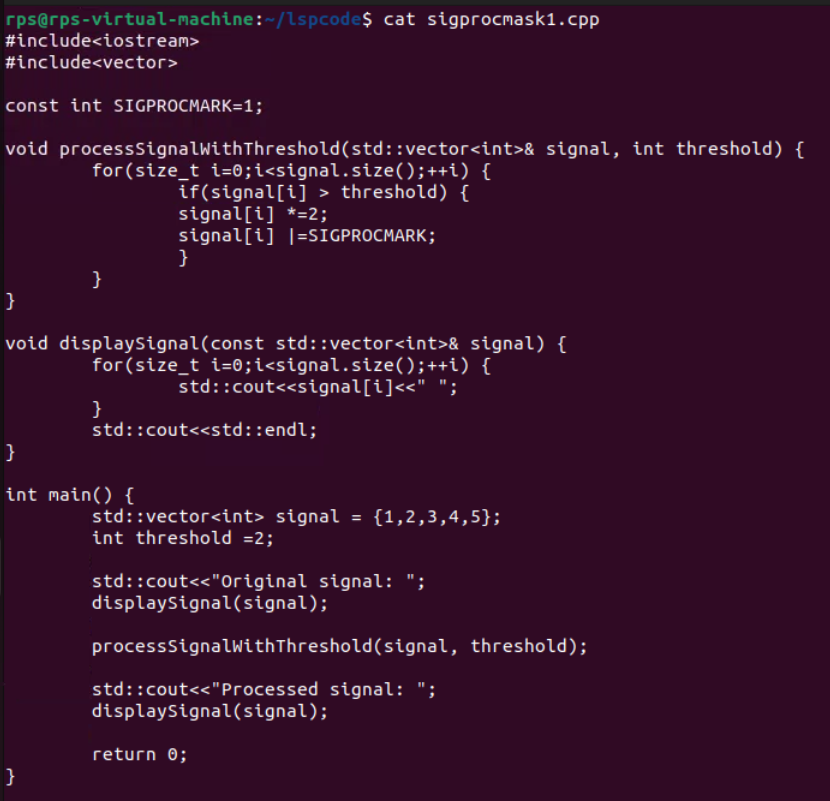
Demonstrate the usage of these functions in a main function with an example signal and a threshold value.

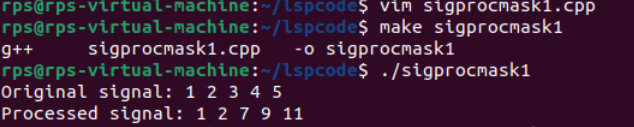
Requirements:

The marker value should be defined as a constant.

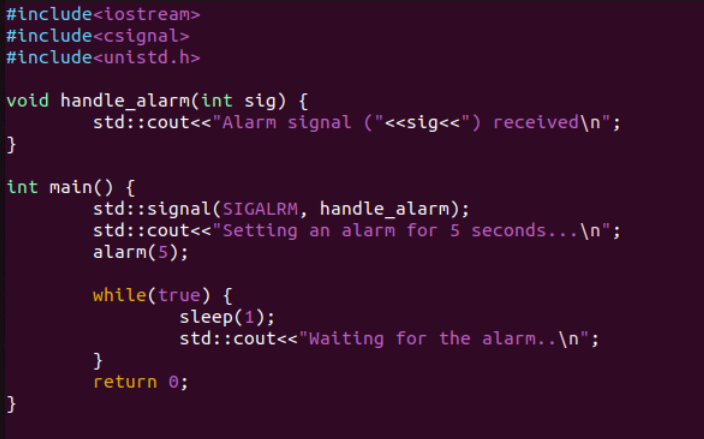
The processSignalWithThreshold function should double the value of each element that exceeds the threshold and use bitwise operations to mark the processed elements.

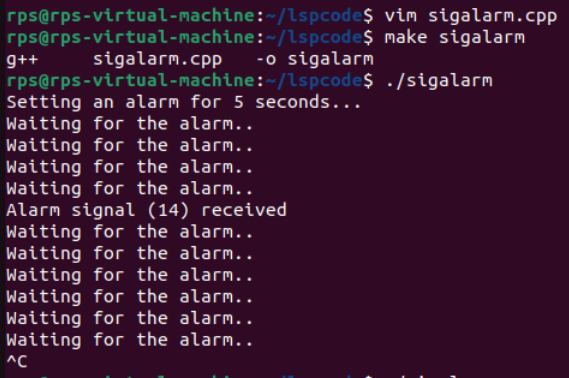
The displaySignal function should print the signal values separated by spaces.





Signal Alarm





Develop a C++ application that demonstrates effective signal handling using SIGALRM, SIGDEFAULT, and SIG\_IGN. The program should:

Set up a timer using alarm() to generate a SIGALRM signal after a specified interval.

Define a signal handler function to process the SIGALRM signal and perform specific actions, such as printing a message, updating a counter, or triggering an event.

Implement logic to handle other signals (e.g., SIGINT, SIGTERM) using SIGDEFAULT or SIG\_IGN as appropriate.

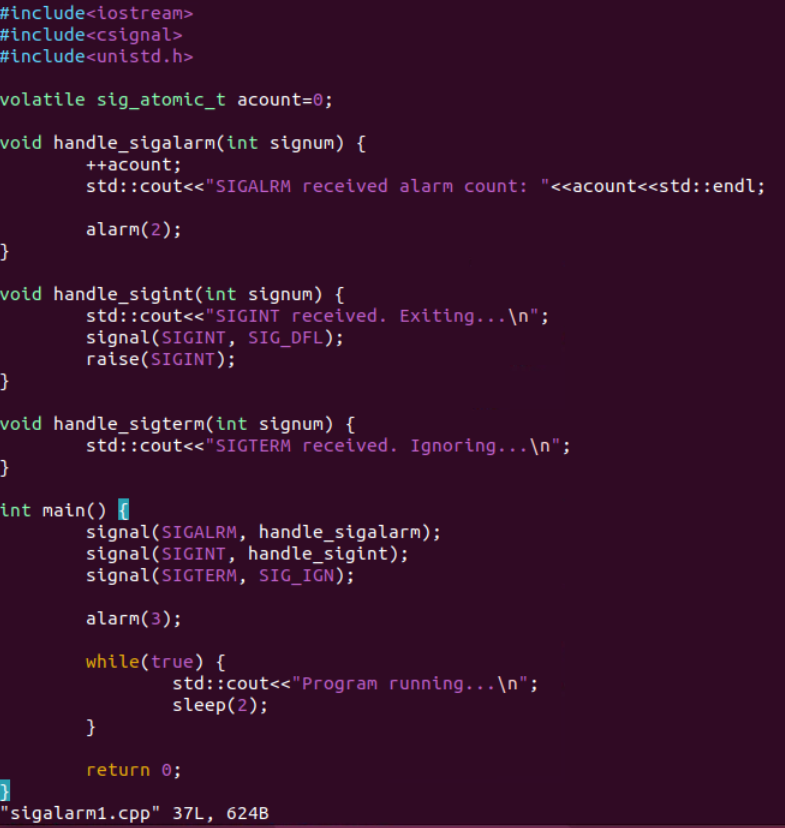
Explore the behavior of the application under different signal combinations and handling strategies.

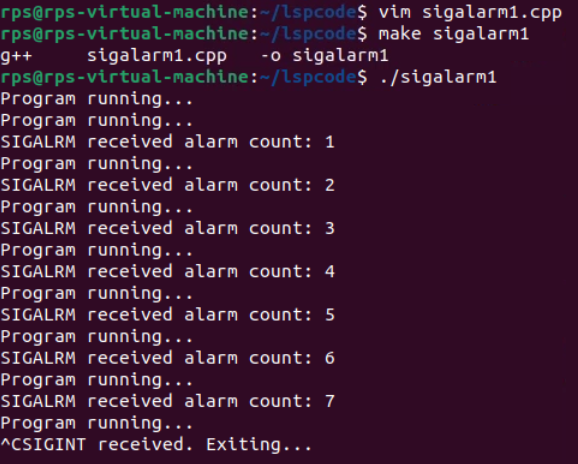
Additional Considerations:

Consider the impact of signal handling on program execution and potential race conditions.

Investigate the use of sigaction for more advanced signal handling capabilities.

Explore the application of signal handling in real-world scenarios, such as timeouts, asynchronous events, and error handling.





SOCKETS

Server code:

#include <iostream>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <cstring>

#define PORT 8080

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

const char \*hello = "Hello from server";

// Creating socket file descriptor

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

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port 8080

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

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

// Forcefully attaching socket to the port 8080

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

perror("bind failed");

exit(EXIT\_FAILURE);

}

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

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket = accept(server\_fd, (struct sockaddr \* )&address, (socklen\_t\*)&addrlen)) < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

read(new\_socket, buffer, 1024);

std::cout << "Message from client: " << buffer << std::endl;

send(new\_socket, hello, strlen(hello), 0);

std::cout << "Hello message sent\n";

close(new\_socket);

close(server\_fd);

return 0;

}

Client code:

#include <iostream>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <cstring>

#define PORT 8080

int main() {

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

const char \*hello = "Hello from client";

char buffer[1024] = {0};

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

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

return -1;

}

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

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

return -1;

}

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

std::cout << "Hello message sent\n";

valread = read(sock, buffer, 1024);

std::cout << "Message from server: " << buffer << std::endl;

close(sock);

return 0;

}

Output:

