DATE :- 24-07-2024

PROGRAM FOR PROCESS_SIGNAL :-

```
rps@rps-virtual-machine:~$ vim process_signal.cpp
rps@rps-virtual-machine:~$ g++ process_signal.cpp -o process_signal
rps@rps-virtual-machine:~$ ./process_signal
Original signal:12345
Processed signal:357911
```

```cpp
#include<iostream>
#include<vector>

const int SIGPROCMARK = 1;

void processSignal(std::vector<int>&signal) {
        for(size_t i=0; i<signal.size(); ++i) {
                signal[i]*= 2;
                signal[i]|= SIGPROCMARK;
        }
}

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

        std::cout << "Original signal:";
        displaySignal(signal);

        processSignal(signal);

        std::cout << "Processed signal:";
        displaySignal(signal);

        return 0;
}
~
```

1task 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:

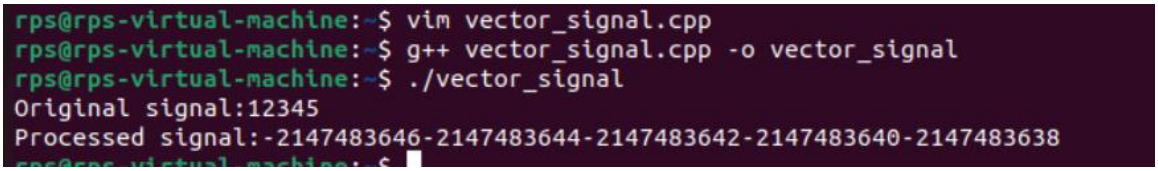
The marker value should be defined as a constant.

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}.



```cpp
#include<iostream>
#include<vector>

const int SIGPROCMARK = 1 << 31;

void processSignal(std::vector<int>& signal) {
        for(int& value:signal) {
                value = (value * 2) | SIGPROCMARK;
        }
}

void displaySignal(const std::vector<int>& signal) {
        for(const int& value:signal) {
                std::cout << value << "";
        }
        std::cout << std::endl;
}

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

        std::cout << "Original signal:";
        displaySignal(signal);

        processSignal(signal);

        std::cout << "Processed signal:";
        displaySignal(signal);

        return 0;
}
```

2. 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.

```
rps@rps-virtual-machine:~$ vim signal_threshold.cpp
rps@rps-virtual-machine:~$ g++ signal_threshold.cpp -o signal_threshold
rps@rps-virtual-machine:~$ ./signal_threshold
Original signal:12345
Processed signal with threshold3:123-2147483640-2147483638
rps@rps-virtual-machine:~$ cat signal_threshold.cpp
#include<iostream>
#include<vector>

const int SIGPROCMARK = 1 << 31;
```

```
#include<iostream>
#include<vector>

const int SIGPROCMARK = 1 << 31;

void processSignalWithThreshold(std::vector<int>& signal, int threshold) {
        for(int& value : signal) {
                if(vaule > threshold) {
                        value = (value*2)|SIGPROCMARK;
                }
        }
}

void displaySignal(const std::vector<int>& signal) {
        for(const int& value : signal) {
                std::cout << value << "";
        }
        std::cout << std::endl;
}

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

        std::cout << "Original signal:";
        displaySignal(signal);

        processSignalWithThreshold(signal, threshold);

        std::cout << "Processed signal with threshold" << threshold << ":";
        displaySinal(signal);

        return 0;
}

-- INSERT -- W10: Warning: Changing a readonly file
```

PROGRAM FOR ALARM :-

```
rps@rps-virtual-machine:~$ vim sig_alarm.cpp
rps@rps-virtual-machine:~$ g++ sig_alarm.cpp -o sig_alarm
rps@rps-virtual-machine:~$ ./sig_alarm
Setting an alarm for 5 seconds...
Waiting for the alarm...
Waiting for the alarm...
Waiting for the alarm...
Waiting for the alarm...
Alarm clock
rps@rps-virtual-machine:~$ cat sig_alarm.cpp
#include<iostream>
#include<csignal>
#include<unistd.h>

void handle_alarm(int sig) {
        std::cout << "Alarm signal (" << sig << ") received.\n";
}

int main() {
        std::signal(SIGINT, handle_alarm);

        std::cout << "Setting an alarm for 5 seconds...\n";
        alarm(5);

        while(true) {
                sleep(1);
                std::cout << "Waiting for the alarm...\n";
        }

        return 0;
}
```

```cpp
#include<iostream>
#include<csignal>
#include<unistd.h>

void handle_alarm(int sig) {
        std::cout << "Alarm signal (" << sig << ") received.\n";
}

int main() {
        std::signal(SIGINT, handle_alarm);

        std::cout << "Setting an alarm for 5 seconds...\n";
        alarm(5);

        while(true) {
                sleep(1);
                std::cout << "Waiting for the alarm...\n";
        }

        return 0;
}
```

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.

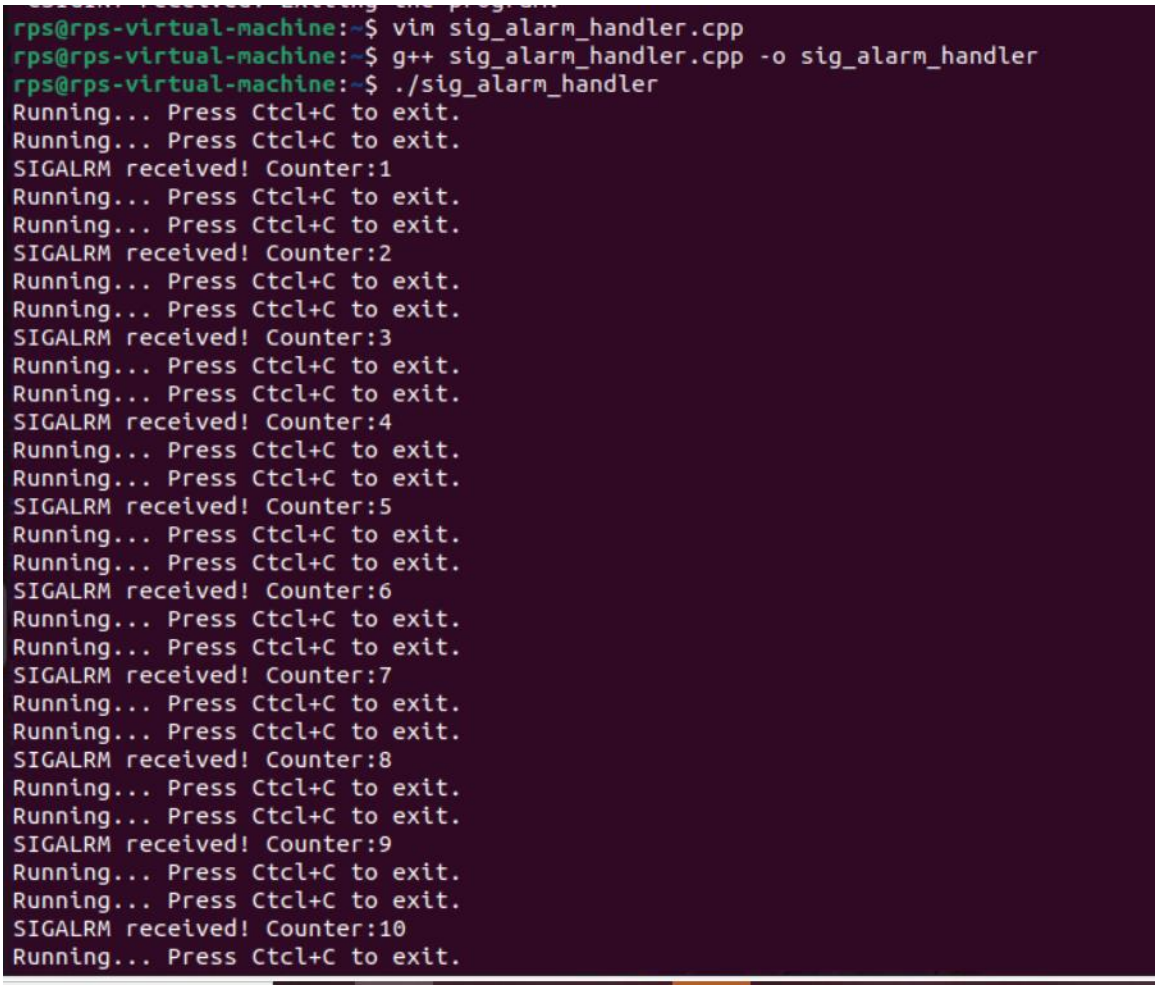
Explore the behavior of the application under different signal combinations and handling strategies.

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.


rps@rps-virtual-machine:~$ vim sig_alarm_handler.cpp
rps@rps-virtual-machine:~$ g++ sig_alarm_handler.cpp -o sig_alarm_handler
rps@rps-virtual-machine:~$ ./sig_alarm_handler
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:1
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:2
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:3
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:4
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:5
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:6
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:7
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:8
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:9
Running... Press Ctcl+C to exit.
Running... Press Ctcl+C to exit.
SIGALRM received! Counter:10
Running... Press Ctcl+C to exit.

```cpp
#include<iostream>
#include<csignal>
#include<unistd.h>

volatile sig_atomic_t alarm_counter = 0;

void handle_sigalrm(int sig) {
       if(sig == SIGALRM) {
          alarm_counter++;
          std::cout << "SIGALRM received! Counter:" << alarm_counter << std::endl;
  alarm(2);
       }
}

void handle_sigint(int sig) {
        if(sig == SIGINT) {
                std::cout << "SIGINT received! Exiting the program." << std::endl;
                exit(0);
        }
}

void setup_signal_handlers() {
        struct sigaction sa;

        sa.sa_handler = handle_sigalrm;
        sa.sa_flags = 0;
        sigemptyset(&sa.sa_mask);
        sigaction(SIGALRM, &sa, nullptr);

        sa.sa_handler = handle_sigint;
        sigaction(SIGINT, &sa, nullptr);

        sa.sa_handler = SIG_IGN;
        sigaction(SIGTERM, &sa, nullptr);
}
```

```
void setup_signal_handlers() {
        struct sigaction sa;

        sa.sa_handler = handle_sigalrm;
        sa.sa_flags = 0;
        sigemptyset(&sa.sa_mask);
        sigaction(SIGALRM, &sa, nullptr);

        sa.sa_handler = handle_sigint;
        sigaction(SIGINT, &sa, nullptr);

        sa.sa_handler = SIG_IGN;
        sigaction(SIGTERM, &sa, nullptr);
}

int main() {
        setup_signal_handlers();

        alarm(2);

        while(true) {
                std::cout << "Running... Press Ctcl+C to exit." << std::endl;
                sleep(1);
        }

        return 0;
}
-- INSERT --
```

PROGRAM FOR SERVER :-

```
rps@rps-virtual-machine:~$ vim socket.cpp
rps@rps-virtual-machine:~$ make socket
g++     socket.cpp    -o socket
rps@rps-virtual-machine:~$ ./socket
accept: Invalid argument
rps@rps-virtual-machine:~$ cat socket.cpp
#include<iostream>
#include<sys/types.h>
#include<netinet/in.h>
```

```
#include<iostream>
#include<sys/types.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";

        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");
                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");
                exit(EXIT_FAILURE);
        }
        if(listen(server_fd, 3) < 0) {
                perror("listen");
```

```
        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);

        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;
}
```

PROGRAM FOR CLIENT :-

```
rps@rps-virtual-machine:~$ vim socket.cpp
rps@rps-virtual-machine:~$ vim server.cpp
rps@rps-virtual-machine:~$ g++ server.cpp -o server
rps@rps-virtual-machine:~$ ./server
Connection Failed
rps@rps-virtual-machine:~$
```

```cpp
#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);

        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;
}
```