DATE :- 23-07-2024

Basic Signal Handling

TASK 1 : Simple Signal Handler:

Write a C++ program that handles the SIGINT signal (Ctrl+C) gracefully by printing a custom message before exiting.

```
rps@rps-virtual-machine:~$ vim simple_signal_handler.cpp
rps@rps-virtual-machine:~$ g++ simple_signal_handler.cpp -o simple_signal_handler
rps@rps-virtual-machine:~$ ./simple_signal_handler
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
program running...
```

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

void signalHandler(int signum) {
        std::cout << "Interrupt signal (" << signum << ") received.\n";
        exit(signum);
}

int main() {

        signal(SIGINT, signalHandler);
        while (1) {
                std::cout << "program running..." << std::endl;
                sleep(1);
        }
        return 0;
}
```

TASK 2: Multiple Signal Handling:

Create a program that handles both SIGINT and SIGTERM signals, printing a different message for each.

```
rps@rps-virtual-machine:~$ vim multiple_signal_handler.cpp
rps@rps-virtual-machine:~$ g++ multiple_signal_handler.cpp -o multiple_signal_handler
rps@rps-virtual-machine:~$ ./multiple_signal_handler
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
```

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

void signalHandler(int signum) {
        if (signum == SIGINT) {
                std::cout << "Interrupt signal (SIGINT) received.\n";
        } else if (signum == SIGTERM) {
                std::cout << "Termination signal (SIGTERM) received.\n";
        }
        exit(signum);
}

int main() {

        signal(SIGINT, signalHandler);
        signal(SIGTERM, signalHandler);

        while(1) {
                std::cout << "Program running..." << std::endl;
                sleep(1);
        }
        return 0;
}
```

```
rps@rps-virtual-machine:~$ vim multiple_signal_handler.cpp
rps@rps-virtual-machine:~$ ps aux | grep multiple_handler
rps       592904  0.0  0.0   9212  2560 pts/1    S+   11:00   0:00 grep --color=auto multiple_handler
rps@rps-virtual-machine:~$ kill -SIGTERM <592904.
bash: 592904.: No such file or directory
rps@rps-virtual-machine:~$ kill -SIGTERM <9212>
bash: syntax error near unexpected token `9212'
rps@rps-virtual-machine:~$ kill -SIGTERM 9212
bash: kill: (9212) - No such process
rps@rps-virtual-machine:~$
```

TASK3: Ignoring Signals:-

Develop a program that ignores the SIGTERM signal and continues execution even after it's sent.

```
rps@rps-virtual-machine:~$ vim ignore_signal.cpp
rps@rps-virtual-machine:~$ g++ ignore_signal.cpp -o ignore_signal
rps@rps-virtual-machine:~$ ./ignore_signal
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
Program running...
```

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

void signalHandler(int signum) {
        std::cout << "signal (" << signum << ") received, but ignored.\n";
}

int main() {

        signal(SIGTERM, SIG_IGN);

        while(1) {
                std::cout << "Program running..." << std::endl;
                sleep(1);
        }
        return 0;
}
~
~
```

```
rps@rps-virtual-machine:~$ ps aux | grep ignore_signal
rps         593092  0.0  0.0   6056  3200 pts/1    S+   11:10   0:00 ./ignore_signal
rps         593131  0.0  0.0   9080  2560 pts/3    S+   11:12   0:00 grep --color=auto ignore_signal
rps@rps-virtual-machine:~$ kill -SIGTERM 8
bash: kill: (8) - Operation not permitted
rps@rps-virtual-machine:~$ kill -SIGTERM 593092
rps@rps-virtual-machine:~$ kill -SIGTERM 593131
bash: kill: (593131) - No such process
rps@rps-virtual-machine:~$
```

PROGRAM FOR SIG_MASK :-

```
rps@rps-virtual-machine:~$ vim sig_mask.cpp
rps@rps-virtual-machine:~$ make sig_mask
g++     sig_mask.cpp    -o sig_mask
rps@rps-virtual-machine:~$ ./sig_mask
SIGINT is blocked for 10 seconds. Try pressing Ctrl+C...
SIGINT is unblocked. Try pressing Ctrl+C again...
```

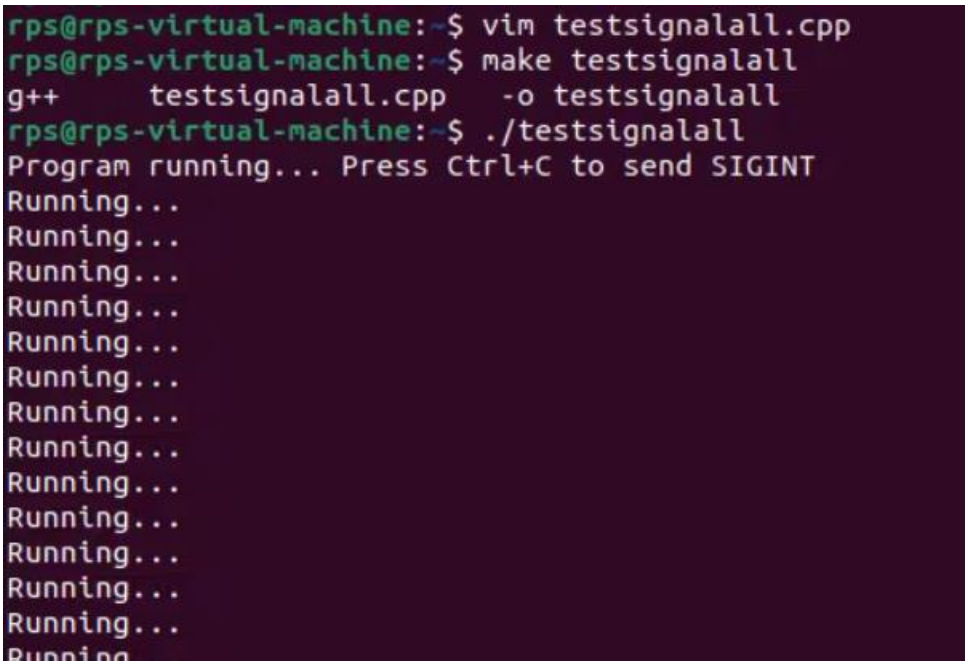
```
rps@rps-virtual-machine:~$ cat sig_mask.cpp
#include<iostream>
#include<csignal>
#include<unistd.h>

void signalHandler(int signum) {
        std::cout << "Interrupt signal (" << signum << ") received.\n";
}

int main() {
        signal(SIGINT, signalHandler);

        sigset_t sigSet;
        sigemptyset(&sigSet);
        sigaddset(&sigSet, SIGINT);

        if(sigprocmask(SIG_BLOCK, &sigSet, nullptr) != 0) {
                std::cerr << "Failed to block SIGINT\n";
                return 1;
        }
        std::cout << "SIGINT is blocked for 10 seconds. Try pressing Ctrl+C...\n";

        sleep(10);

        if(sigprocmask(SIG_UNBLOCK, &sigSet, nullptr) != 0) {
                std::cerr << "Failed to unblock SIGINT\n";
                return 1;
        }
        std::cout << "SIGINT is unblocked. Try pressing Ctrl+C again...\n";
        sleep(10);
        return 0;
}
rps@rps-virtual-machine:~$
```

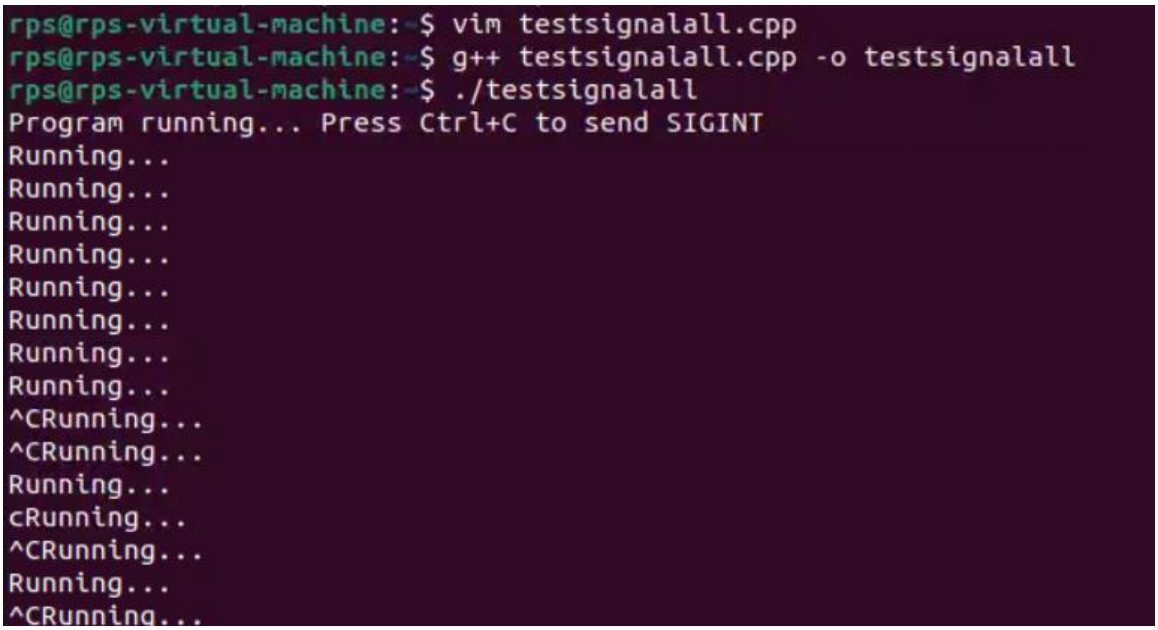
PROGRAM FOR SIG_MASK1 :-

```
rps@rps-virtual-machine:~$ vim sig_mask1.cpp
rps@rps-virtual-machine:~$ make sig_mask1
g++     sig_mask1.cpp   -o sig_mask1
rps@rps-virtual-machine:~$ ./sig_mask1
SIGTERM is blocked during critical section.
Entering critical section...
Exiting critical section...
SIGTERM is unblocked.
rps@rps-virtual-machine:~$ cat sig_mask1.cpp
```

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

void signalHandler(int signum) {
        std::cout << "Signal (" << signum << ") received.\n";
}

int main() {
        signal(SIGTERM, signalHandler);

        sigset_t sigSet;
        sigemptyset(&sigSet);
        sigaddset(&sigSet, SIGTERM);

        if(sigprocmask(SIG_BLOCK, &sigSet, nullptr) != 0) {
                std::cerr << "Failedto block SIGTERM\n";
                return 1;
        }
        std::cout << "SIGTERM is blocked during critical section.\n";

        std::cout << "Entering critical section...\n";
        sleep(5);
        std::cout << "Exiting critical section...\n";

        if (sigprocmask(SIG_UNBLOCK, &sigSet, nullptr) != 0) {
                std::cerr << "Failed to unblock SIGTERM\n";
                return 1;
        }
        std::cout << "SIGTERM is unblocked.\n";
        sleep(10);
        return 0;
}
rps@rps-virtual-machine:~$
```

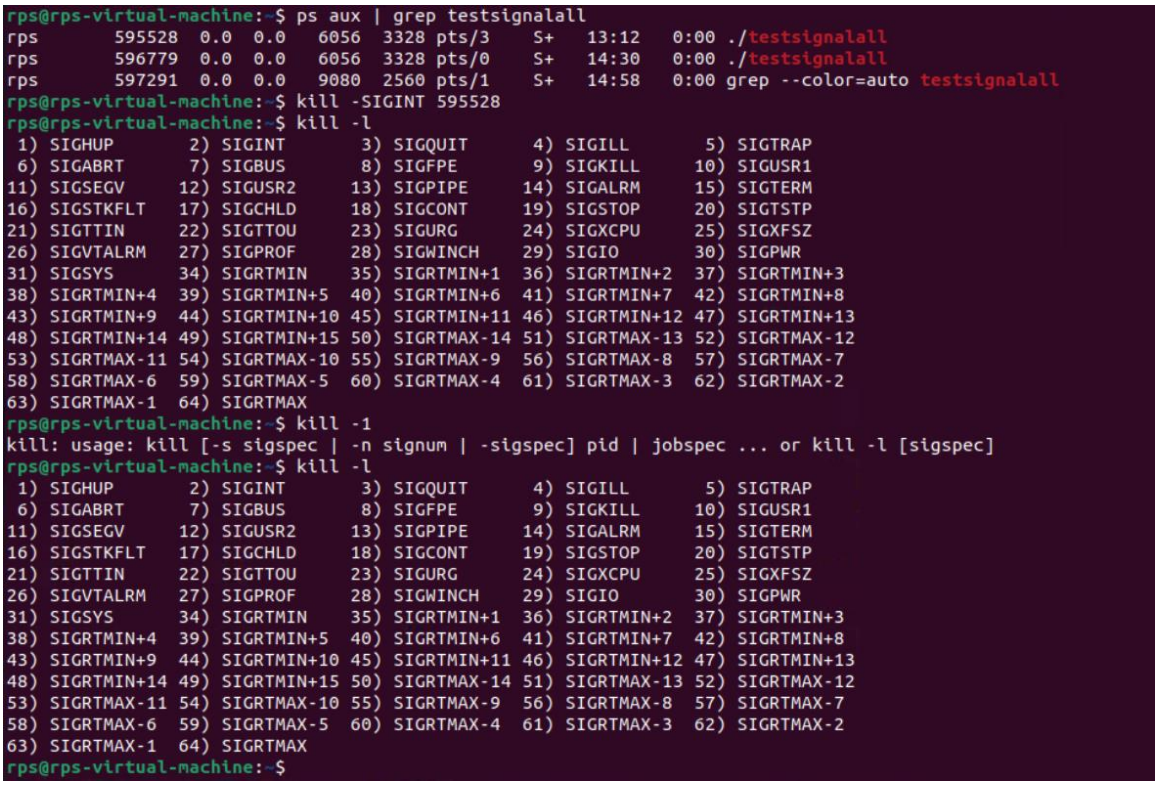
PROGRAM FOR TESTSIGNALALL :-


rps@rps-virtual-machine:~$ vim testsignalall.cpp
rps@rps-virtual-machine:~$ make testsignalall
g++ testsignalall.cpp -o testsignalall
rps@rps-virtual-machine:~$ ./testsignalall
Program running... Press Ctrl+C to send SIGINT
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...


rps@rps-virtual-machine:~$ vim testsignalall.cpp
rps@rps-virtual-machine:~$ g++ testsignalall.cpp -o testsignalall
rps@rps-virtual-machine:~$ ./testsignalall
Program running... Press Ctrl+C to send SIGINT
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
^CRunning...
^CRunning...
Running...
cRunning...
^CRunning...
Running...
^CRunning...

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

void myHandler(int signum) {
        std::cout << "Handled signal(" << signum << "): " << strsignal(signum) << std::endl;
}

int main() {
        void (*pRet)(int);

        pRet = signal(SIGHUP, myHandler);
        pRet = signal(SIGINT, myHandler);
        pRet = signal(SIGQUIT, myHandler);
        pRet = signal(SIGILL, myHandler);
        pRet = signal(SIGTRAP, myHandler);
        pRet = signal(SIGABRT, myHandler);
        pRet = signal(SIGBUS, myHandler);
        pRet = signal(SIGFPE, myHandler);
        pRet = signal(SIGKILL, myHandler);

        sigset_t signalSet;
        sigemptyset(&signalSet);

        sigaddset(&signalSet, SIGINT);
        sigaddset(&signalSet, SIGQUIT);

        sigprocmask(SIG_BLOCK, &signalSet, nullptr);

        std::cout << "Program running... Press Ctrl+C to send SIGINT" << std::endl;
```

```cpp
int main() {
        void (*pRet)(int);

        pRet = signal(SIGHUP, myHandler);
        pRet = signal(SIGINT, myHandler);
        pRet = signal(SIGQUIT, myHandler);
        pRet = signal(SIGILL, myHandler);
        pRet = signal(SIGTRAP, myHandler);
        pRet = signal(SIGABRT, myHandler);
        pRet = signal(SIGBUS, myHandler);
        pRet = signal(SIGFPE, myHandler);
        pRet = signal(SIGKILL, myHandler);

        sigset_t signalSet;
        sigemptyset(&signalSet);

        sigaddset(&signalSet, SIGINT);
        sigaddset(&signalSet, SIGQUIT);

        sigprocmask(SIG_BLOCK, &signalSet, nullptr);

        std::cout << "Program running... Press Ctrl+C to send SIGINT" << std::endl;

        while(1) {
                std::cout << "Running..." << std::endl;
                sleep(1);
        }
        return 0;
}
```



Problem Statement 2: Signal Masking and Unmasking for Graceful Shutdown

Problem: Develop a C++ application that gracefully handles termination signals (e.g., SIGTERM, SIGINT) by masking specific signals during critical operations and unmasking them afterwards. Implement a clean shutdown procedure that ensures all resources are released before the process exits.

Key Challenges:

Determining the appropriate signals to mask during critical operations.

Ensuring timely unmasking of signals to avoid process hangs.

Implementing a robust shutdown mechanism that handles unexpected interruptions.

```
rps@rps-virtual-machine:~$ vim graceful_shutdown.cpp
rps@rps-virtual-machine:~$ make graceful_shutdown
g++     graceful_shutdown.cpp   -o graceful_shutdown
rps@rps-virtual-machine:~$ ./graceful_shutdown
Program running... Press Ctrl+C or send SIGTERM to initiate shutdown.
Starting critical operation...
Finished critical opeartion...
Starting critical operation...
Finished critical opeartion...
Starting critical operation...
Finished critical opeartion...
Starting critical operation...
Finished critical opeartion...
Starting critical operation...
Finished critical opeartion...
Starting critical operation...
^C
Finished critical opeartion...
Received signal (2): Interrupt
Performing clean shutdown...
Clean shutdown complete. Exiting program.
```

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

std::atomic<bool> shutdownInitiated(false);

void signalHandler(int signum) {
        std::cout << "Received signal ("<< signum <<"): "<< strsignal(signum) << std::endl;
        if(signum == SIGTERM || signum == SIGINT) {
                shutdownInitiated = true;
        }
}

void criticalOperation() {
        sigset_t newSet, oldSet;
        sigemptyset(&newSet);
        sigaddset(&newSet, SIGTERM);
        sigaddset(&newSet, SIGINT);

        sigprocmask(SIG_BLOCK, &newSet, &oldSet);

        std::cout << "Starting critical operation..." << std::endl;
        sleep(5);
        std::cout << "Finished critical opeartion..." << std::endl;

        sigprocmask(SIG_SETMASK, &oldSet, nullptr);
}

void cleanShutdown() {
        std::cout << "Performing clean shutdown..." << std::endl;
        std::cout << "Clean shutdown complete. Exiting program." << std::endl;
}
```

```
void cleanShutdown() {
        std::cout << "Performing clean shutdown..." << std::endl;
        std::cout << "Clean shutdown complete. Exiting program." << std::endl;
}
int main() {
        signal(SIGTERM, signalHandler);
        signal(SIGINT, signalHandler);

        std::cout << "Program running... Press Ctrl+C or send SIGTERM to initiate shutdown." << std::endl;
        while (!shutdownInitiated) {
                criticalOperation();
                sleep(1);
        }
        cleanShutdown();
        return 0;
}
```

PROGRAM SIGNAL_FILE_HANDLING :-

```
rps@rps-virtual-machine:~$ vim signal_file_handling.cpp
rps@rps-virtual-machine:~$ g++ signal_file_handling.cpp -o signal_file_handling
rps@rps-virtual-machine:~$ ./signal_file_handling
File tempfile.txt created.
Press Ctrl+C to trigger the signal handler...
^C
Interrupt signal (2) received.
Filetempfile.txt deleted successfully.
rps@rps-virtual-machine:~$
```

```cpp
#include<iostream>
#include<csignal>
#include<stdlib.h>
#include<cstdio>
#include<chrono>
#include<thread>

const char* filename = "tempfile.txt";

void signalHandler(int signum) {
        std::cout << "\nInterrupt signal (" << signum << ") received.\n";

        if(std::remove(filename) == 0) {
                std::cout << "File" << filename << " deleted successfully.\n";
        } else {
                std::perror("Error deleting file");
        }
        exit(signum);
}

int main() {
        FILE* file = std::fopen(filename, "w");
        if (file == nullptr) {
                std::perror("Error creating file");
                return 1;
        }
        std::fputs("Temporary file content.", file);
        std::fclose(file);
        std::cout << "File " << filename << " created.\n";

        std::signal(SIGINT, signalHandler);

        std::cout << "Press Ctrl+C to trigger the signal handler...\n";
```

```cpp
int main() {
        FILE* file = std::fopen(filename, "w");
        if (file == nullptr) {
                std::perror("Error creating file");
                return 1;
        }
        std::fputs("Temporary file content.", file);
        std::fclose(file);
        std::cout << "File " << filename << " created.\n";

        std::signal(SIGINT, signalHandler);

        std::cout << "Press Ctrl+C to trigger the signal handler...\n";

        while (true) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        return 0;
}
```

PROGRAM FOR CAUGHT_SIGNAL :-

```
rps@rps-virtual-machine:~$ vim caught_signal.cpp
rps@rps-virtual-machine:~$ g++ caught_signal.cpp -o caught_signal
rps@rps-virtual-machine:~$ ./caught_signal
Press Ctrl+C to generate SIGINT signal,
Waiting for signal...
Ignoring SIGINT signal for the next 10 seconds.
Restoring default handling for SIGINT signal.
rps@rps-virtual-machine:~$
```

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

void signalHandler(int signum) {
        std::cout << "Caught signal " << signum << std::endl;
}

int main() {
        signal(SIGINT, signalHandler);
        std::cout << "Press Ctrl+C to generate SIGINT signal," << std::endl;
        std::cout << "Waiting for signal..." << std::endl;

        sleep(10);

        ut << "Ignoring SIGINT signal for the next 10 seconds." << std::endl;
        signal(SIGINT, SIG_IGN);

        sleep(10);

        std::cout << "Restoring default handling for SIGINT signal." << std::endl;
        signal(SIGINT, SIG_DFL);

        sleep(10);
        return 0;
}
```