Lab Manual 9
(Operating Systems)
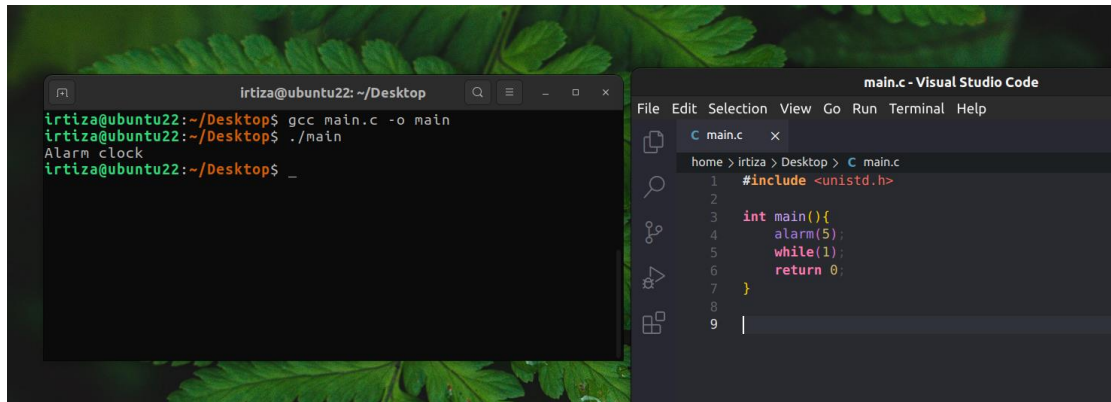
Department of Computer Science
FAST-NU, Lahore

Signals

- A signal is a software notification, of an event, to a process, and it is generated when the event that causes the signal occurs.
- A signal is considered delivered when the process takes an action based on the type of the signal.
- A signal that has been generated but not yet delivered, is said to be pending.
- The process must be in running phases at the time of signal delivery.
- A process can *catch* the signal by executing a signal handler. A signal handler is installed by calling **signalaction** with the name of user written function.
- Signal are asynchronous to the process. Which means the process does not have to call or be at the location of signal handler to handle/catch the signal.

signal	description	default action
SIGABRT	process abort	implementation dependent
SIGALRM	alarm clock	abnormal termination
SIGBUS	access undefined part of memory object	implementation dependent
SIGCHLD	child terminated, stopped or continued	ignore
SIGCONT	execution continued if stopped	continue
SIGFPE	error in arithmetic operation as in division by zero	implementation dependent
SIGHUP	hang-up (death) on controlling terminal (process)	abnormal termination
SIGILL	invalid hardware instruction	implementation dependent
SIGINT	interactive attention signal (usually Ctrl-C)	abnormal termination
SIGKILL	terminated (cannot be caught or ignored)	abnormal termination
SIGPIPE	write on a pipe with no readers	abnormal termination
SIGQUIT	interactive termination: core dump (usually Ctrl-|)	implementation dependent
SIGSEGV	invalid memory reference	implementation dependent
SIGSTOP	execution stopped (cannot be caught or ignored)	stop
SIGTERM	termination	abnormal termination
SIGTSTP	terminal stop	stop
SIGTTIN	background process attempting to read	stop
SIGTTOU	background process attempting to write	stop
SIGURG	high bandwidth data available at a socket	ignore
SIGUSR1	user-defined signal 1	abnormal termination
SIGUSR2	user-defined signal 2	abnormal termination

Example: Abnormal termination of program with SIGALRM

Here the alarm(5) generates SIGALRM after 5 seconds and the program terminates abnormally with the message **Alarm Clock.** An infinite while loop was added to keep the program in running state.

Handling Signals

We can also define our own signals handlers by calling **signal()** system call and passing the signal identifier and the handler function.

Example: Generating **SIGINT** i.e. signal for interrupt (e.g. when CTRL + C is pressed).

Remember how when you click CTRL + C the program terminates ? But here I have changed the signal handler and have told the program to just print a message rather than exiting. Hence whenever I press CTRL + C the program just prints a message and keeps on running.

How to terminate this process now?

By sending some other signal that causes the termination of a process.

Here I sent SIGTERM signal to the main process, which had a PID of 3881. I did this from another process i.e. shell. The **–s** used in the sending of signal stands for signal, which is followed by the type of signal we want to send to the process with following PID.

Note that *kill* command is used to send a signal to a process. It doesn't simply kill/terminate a process, but rather the signal sent to the process kills/terminates it.

```
irtiza@ubuntu22:~/Desktop$ gcc main.c -o main
irtiza@ubuntu22:~/Desktop$ ./main
^CI have handled the interrupt successfully
^CI have handled the interrupt successfully
^CI have handled the interrupt successfully
Terminated
irtiza@ubuntu22:~/Desktop$ _
```
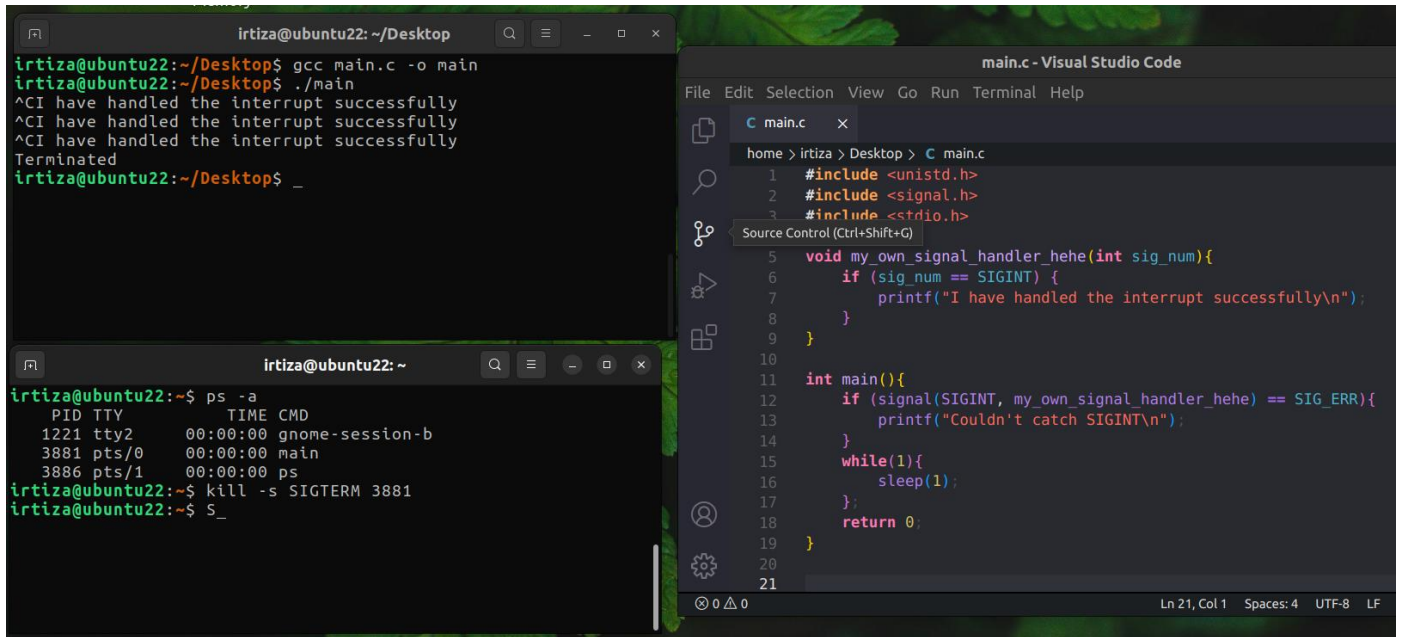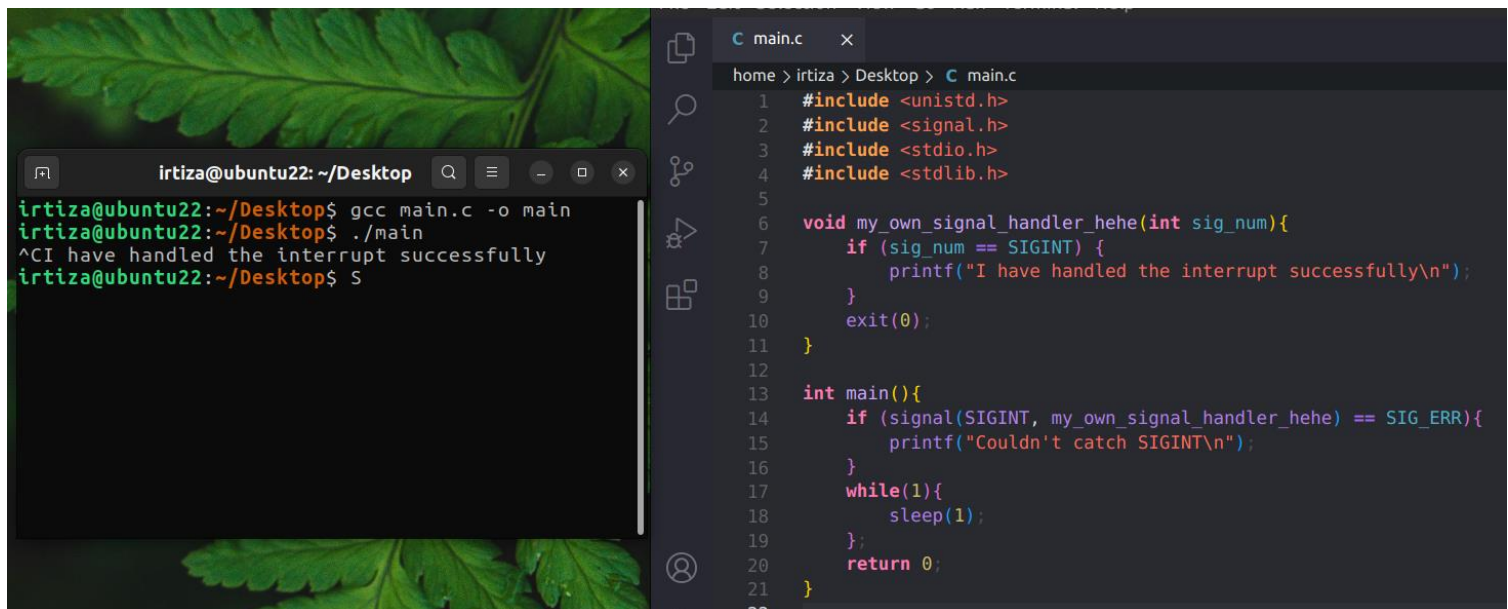
```
irtiza@ubuntu22:~$ ps -a
   PID TTY          TIME CMD
  1221 tty2     00:00:00 gnome-session-b
  3881 pts/0    00:00:00 main
  3886 pts/1    00:00:00 ps
irtiza@ubuntu22:~$ kill -s SIGTERM 3881
irtiza@ubuntu22:~$ S_
```
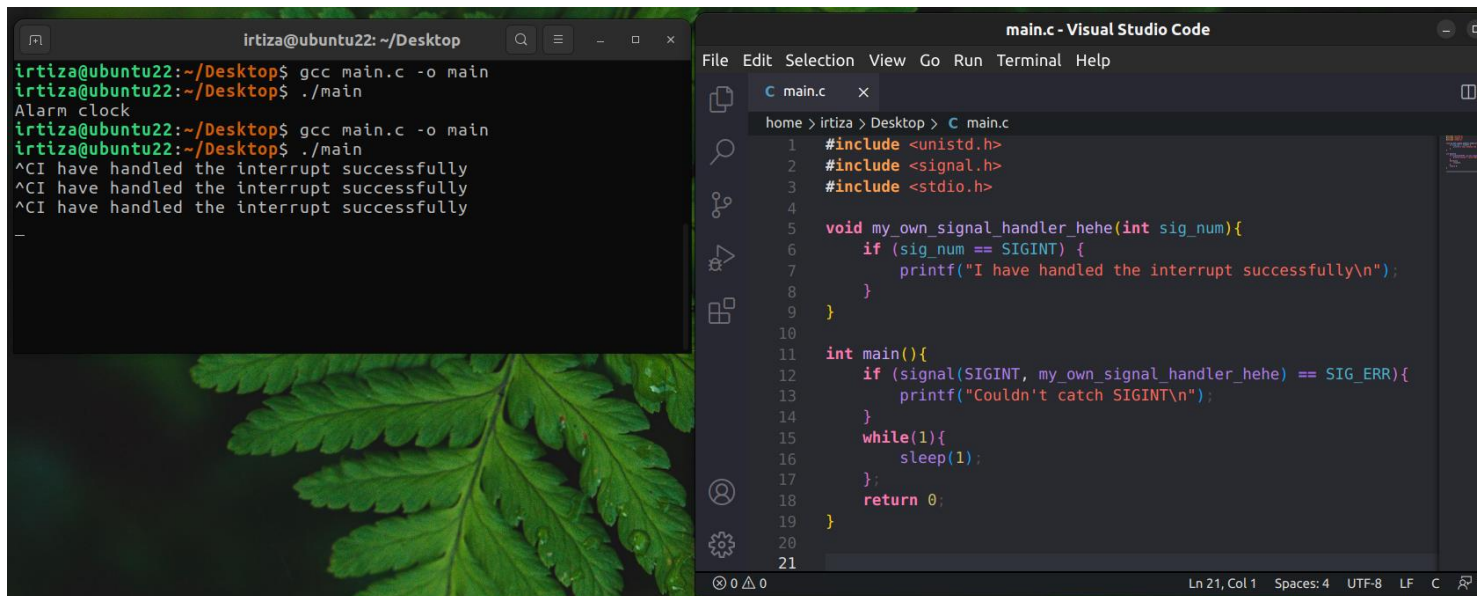
main.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

C main.c

home > irtiza > Desktop > C main.c

```c
1    #include <unistd.h>
2    #include <signal.h>
3    #include <stdio.h>
4
5    void my_own_signal_handler_hehe(int sig_num){
6        if (sig_num == SIGINT) {
7            printf("I have handled the interrupt successfully\n");
8        }
9    }
10
11   int main(){
12       if (signal(SIGINT, my_own_signal_handler_hehe) == SIG_ERR){
13           printf("Couldn't catch SIGINT\n");
14       }
15       while(1){
16           sleep(1);
17       };
18       return 0;
19   }
20
21
```

Ln 21, Col 1   Spaces: 4   UTF-8   LF

Here I have updated my signal handler function and told the program to exit after printing a message.

```
irtiza@ubuntu22:~/Desktop$ gcc main.c -o main
irtiza@ubuntu22:~/Desktop$ ./main
^CI have handled the interrupt successfully
irtiza@ubuntu22:~/Desktop$ S
```

C main.c

home > irtiza > Desktop > C main.c

```c
1    #include <unistd.h>
2    #include <signal.h>
3    #include <stdio.h>
4    #include <stdlib.h>
5
6    void my_own_signal_handler_hehe(int sig_num){
7        if (sig_num == SIGINT) {
8            printf("I have handled the interrupt successfully\n");
9        }
10       exit(0);
11   }
12
13   int main(){
14       if (signal(SIGINT, my_own_signal_handler_hehe) == SIG_ERR){
15           printf("Couldn't catch SIGINT\n");
16       }
17       while(1){
18           sleep(1);
19       };
20       return 0;
21   }
22
```

In Lab Tasks: (10 Marks)

- Recreate the stopwatch application.
- Take the number of laps and lap time from the user.
- You will create a child process, which prints lap-number after the mentioned number of seconds.
- Implement your own signal handler that handles the death of a child signal.
- Once the above-created child is terminated, your program should call another function, via handler, that prints the total time in seconds.
- Also, create a suitable MAKEFILE for the execution of this program.

Note 1: You are to create only one child process for this task.

Output should be:

```
irtiza@ubuntu22:~/Desktop$ gcc main.c -o main
irtiza@ubuntu22:~/Desktop$ ./main
Enter Number of Laps: 5
Enter Lap Time: 3
Lap: 1 Completed
Lap: 2 Completed
Lap: 3 Completed
Lap: 4 Completed
Lap: 5 Completed
Received SIGCHLD
Total Time in Seconds: 15
```

Submit File with name as: YOUR_ROLLNUMBER.c and also submit output with name YOUR_ROLLNUMBER_OUTPUT.jpg