# LAB 14
# Breeha Qasim

## Section 2

```
  GNU nano 6.2                                  loop.c
#include <stdio.h>
#include <unistd.h> // for sleep()

int
main(int argc, char *argv[])
{
    int j;

    for (j = 0; ; j++) {
        printf("%d\n", j);
        sleep(3);
    }
    /* Loop slowly... */
}
```

[ Read 14 lines ]

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ nano loop.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ gcc loop.c -o loop
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ ./loop
0
1
^X^X^C
```

## Section 3

```
 GNU nano 6.2                                  signal0.c *
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for sleep()
#include <signal.h>

int
main(int argc, char *argv[])
{
    int j;
    if (signal(SIGINT, SIG_IGN) == SIG_ERR) {
        fprintf(stderr, "Error ignoring signal\n");
        exit(1);
    }

    for (j = 0; ; j++) {
        printf("%d\n", j);
        sleep(3);
    }
    /* Loop slowly... */
}
```

Question 1:
To terminate it I first passed this command **ps -aux | grep signal0** on a new terminal to get the pid of process.

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx: ~/Documents/b...   hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx: ~/Documents/b...
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ ps -aux | grep signal0
hp          6294  0.0  0.0   2776  1408 pts/0    S+   15:44   0:00 ./signal0
hp          6320  0.0  0.0   9564  2688 pts/1    S+   15:45   0:00 grep --color=auto signal0
```

Then after passing this command **kill -9 6294** it terminates the process.

Question 2:
By giving SIG_IGN as the action, it is possible to ignore the SIGINT signal, which is produced by pressing Ctrl+C. This instructs the operating system to do nothing in response to the signal and to forego its default behavior, which is typically process termination. The program ends and an error message appears if the signal function is unsuccessful.

## Section 4

```
 GNU nano 6.2                                  signal1.c *
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for sleep()
#include <signal.h>

//#include "tlpi_hdr.h"

static int
signal_handler(int sig)
{
    printf("Ouch! Signal %d received\n", sig);
    return 1;
}


int
main(int argc, char *argv[])
{
    int j;
    if (signal(SIGINT, sig_handler) == SIG_ERR) {
        fprintf(stderr, "signal handler not registered\n");
        exit(1);
    }

    for (j = 0; ; j++) {
        printf("%d\n", j);
        sleep(3);
    }
    /* Loop slowly... */
}
```

Question 3:

3.1) What happened?

When I change the void signal_handler function to return a value (e.g. static int signal_handler), compilation warnings are generated,

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ gcc signal1.c -o signal1
signal1.c: In function ‘main’:
signal1.c:20:24: error: ‘sig_handler’ undeclared (first use in this function); did you mean ‘sa_handler’?
   20 |     if (signal(SIGINT, sig_handler) == SIG_ERR) {
      |                        ^~~~~~~~~~~
      |                        sa_handler
signal1.c:20:24: note: each undeclared identifier is reported only once for each function it appears in
```

3.2) Explain why did that happen.

Handlers having the signature void (*)(int) are strictly required by the function. Program and operating system expectations diverge when the return type is changed because it interferes with the asynchronous signal handling procedure.

Question 4:
During execution, I used Ctrl+Z to interrupt the terminal before quit/exit is called. This caused the SIGTSTP signal to be sent, which suspended the process. The process ends with a core dump if a SIGQUIT signal (delivered by Ctrl+\) is sent in its place. This behavior illustrates the impact of several signals on process control.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for sleep()
#include <signal.h>

//#include "tlpi_hdr.h"

static void
sig_handler(int sig)
{
    printf("Ouch!\n");
}

int
main(int argc, char *argv[])
{
    int j;
    if (signal(SIGINT, sig_handler) == SIG_ERR) {
        fprintf(stderr, "signal handler not registered\n");
        exit(1);
    }

    for (j = 0; ; j++) {
        printf("%d\n", j);
        sleep(3);
    }
    /* Loop slowly... */
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ gcc interrupt_terminal.c -o interrupt
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ ./interrupt
0
1
2
^Z
[2]+  Stopped                 ./interrupt
```

## Section 5

```
  GNU nano 6.2                                  exrcise5.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Child process
        printf("First Name: Breeha\n");
    } else { // Parent process
        int delay = 0 + 8; // Sum of first 2 digits of Student ID
        sleep(delay);
        printf("Second Name: Qasim, Student ID: 08283\n");
    }

    return 0;
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ nano exrcise5.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ gcc exrcise5.c -o 5
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab14$ ./5
First Name: Breeha
Second Name: Qasim, Student ID: 08283
```

**Explanation:**

To guarantee that the child process prints first, the parent process's sleep() function delays its execution. This illustrates how delays may be used to synchronize operations and how they can be split up to carry out different tasks.