(2025 OS) 3. Multi-process Programming and Signal Handling

This assignment primarily aims to have everyone practice the programming of multi-process, signals, and exception handling.

Throughout this assignment, you will engage with the following key concepts:

1. **Multi-Process Interaction:** You will employ the [fork()](#) system call to create multiple processes, allowing them to explore the intricacies of parallelism and concurrent execution. This experience will show how processes communicate, share resources, and cooperate to achieve specific objectives.

2. **Process Synchronization:** The assignment's use of [wait()](#) will let you practice synchronization, enabling processes to manage the execution order. Understanding how to coordinate processes is essential in preventing race conditions and ensuring orderly execution.

3. **Signaling Mechanisms:** By utilizing [kill()](#) and [signal()](#), you will learn how to harness the power of signals for inter-process communication and event handling. This aspect of the assignment provides insight into asynchronous communication between processes and demonstrates how signals can be customized to trigger specific actions.

4. **Exception Handling with [longjmp()](#) and [setjmp()](#):** The use of `longjmp()` and `setjmp()` will familiarize you with exception handling mechanisms that facilitate non-local jumps within the program's control flow. Through this, you will learn how to gracefully handle errors and unexpected situations, enhancing the reliability and robustness of their programs.

Assignment

In this assignment, you are required to finish the implementation of two programs `hw3_1` and `hw3_2`.

In `hw3_1,` the parent process will fork a child process. The parent and child processes will then operate on the variables within their own processes. You need to ensure that the child process completes its execution before the parent process. Please complete the missing code

at line 20 so that the output will look as follows. You **should not** modify the rest of the code in hw3_1.

```
trista@trista:~/OS_HW3$ gcc -o hw3_1 hw3_1.c
trista@trista:~/OS_HW3$ ./hw3_1
Child has x = 11
Parent has x = -4
trista@trista:~/OS_HW3$ 
```

In your report, please provide answers to the following questions:

Q1: What are the issues in the original hw3_1 program? Specifically, what would have happened without the code you added at line 20?

Q2: Explain why the values of variable "x" in the parent process and the child process can be different (11 vs -4).

In `hw3_2`, this program will request the user to input a "jump case" number. Subsequently, it will fork a child process.

The child process will keep printing the string 'Child is waiting for signal' on the screen until it receives SIGINT. After receiving a SIGINT, it will move on to wait for a SIGKILL.

The parent process will wait for SIGINT and keep printing 'Parent is waiting for signal' on the screen. Once the parent receives a SIGINT, it will send a SIGKILL to the child to kill the child process.

If the jump case number is '0' or '1', the parent process will terminate the child process with SIGKILL. For the other jump cases, the parent process should print "unknown error," and the child process will remain waiting to be killed. You need to press 'ctrl+z' to stop and kill the child process yourself.

Please complete the missing lines in `hw3_2` and ensure the output looks as follows.

Case1 (when the user enters '0' or '1'):

```
trista@trista:~/OS_HW3$ ./hw3_2
Input jump_case : 1
Parent is waiting for signal
Child is waiting for signal
Parent is waiting for signal
Child is waiting for signal
Parent is waiting for signal
Child is waiting for signal
^C
Child is waiting for kill
Child is waiting for kill
Child is waiting for kill

Going to kill child
Child is waiting for kill
Child is waiting for kill
Child is waiting for kill
Child has been killed.
END
trista@trista:~/OS_HW3$
```

Others (when the user enters the other characters):

```
trista@trista:~/OS_HW3$ ./hw3_2
Input jump_case : 2
Parent is waiting for signal
Child is waiting for signal
Parent is waiting for signal
Child is waiting for signal
Parent is waiting for signal
Child is waiting for signal
^C
Child is waiting for kill

 Unknown Error
Child is waiting for kill

 Unknown Error
Child is waiting for kill

Child is waiting for kill
 Unknown Error
Child is waiting for kill

 Unknown Error
^Z
[1]+  Stopped                 ./hw3_2
trista@trista:~/OS_HW3$
```

```
trista@trista:~/OS_HW3$ jobs
[1]+  Stopped                 ./hw3_2
trista@trista:~/OS_HW3$ kill %1

[1]+  Stopped                 ./hw3_2
trista@trista:~/OS_HW3$ jobs
[1]+  Terminated              ./hw3_2
trista@trista:~/OS_HW3$ jobs
trista@trista:~/OS_HW3$ █
```

Please also provide answers to the following questions in your report:

Q3: Explain the difference between "setjmp & longjmp" and "goto".

- Note that you can send SIGINT to the parent and the child by pressing 'ctrl+c' on the keyboard.
- You can install the signal handler for SIGINT at Line 19. The installed handler will be inherited by the child process after you perform the process fork at Line 20.
- You don't need to install a SIGKILL handler in the child. The C runtime has a default handler for SIGKILL. The default handler will terminate the process after receiving the SIGKILL signal.

Submission

Please submit a **zip** file to E3 containing your **program sources** and **report**.

For the program source code (40%):

- The program must implemented using C.
- Make sure your code can be compiled on **Ubuntu 22.04 AMD64**.
- Make sure your outputs are correct.

For the report part (60%):

- Make sure to include your output screenshots.
- Make sure to answer the above questions Q1, Q2, and Q3.

The name of the zip file should be <student_id>.zip, and the structure of the file should be as the following:

<stduent_id>.zip

   |- <student_id>/

      |- hw3_1.c

```
|- hw3_2.c
|- hw3.pdf
```

If you have any questions, please contact TA 李亭萱 via E3 platform or email <trista.cs12@nycu.edu.tw>