Lab 2: Fork System Call in C (*5 Marks*)

Student Name: *Muhammad 'Asif Danial Bin Mohd Shahir*

Github Repository for Lab 2: *https://github.com/asfdnl/ITT440-Lab2*

Matric No: *2021114813*

Group: *CS2554A*

2.1 Basic Fork Example

Basic knowledge of Fork() usage

Unix Process

- An entity that executes a given piece of code
- has its own execution stack
- has its own set of memory pages
- has its own file descriptors table
- A unique process ID

The fork() System Call

- Basic way to create a new process.
- It is also a unique system call, since it returns twice to the caller.

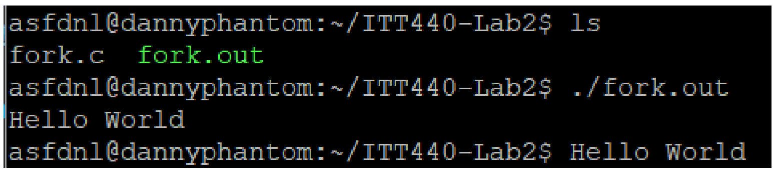
This system call causes the current process to be split into two processes

- a parent process
- a child processes

Please compile and run the following code on your Ubuntu/Linux

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main(){

fork();

printf("Hello World \n");

return 0;

}
```

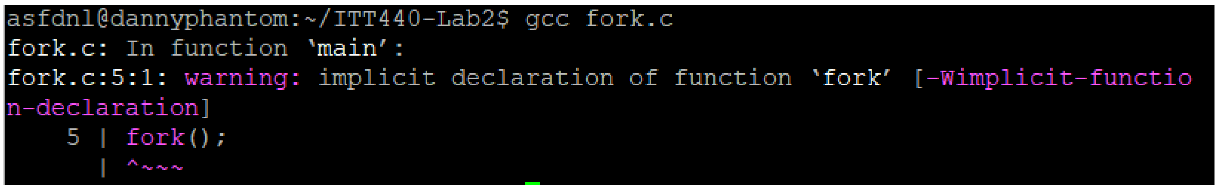
How many times the program print its output? Please provide screenshot. Please briefly explain why **(0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ls
fork.c  fork.out
asfdnl@dannyphantom:~/ITT440-Lab2$ ./fork.out
Hello World
asfdnl@dannyphantom:~/ITT440-Lab2$ Hello World
```

The program prints its output twice because the first output is run by the parent and the second one by the child process.

Based on code given in 2.1 above, please comment out the line #include <unistd.h> and compile and run the code, what is the output, please also provide screenshot. Explain why (**0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ gcc fork.c
fork.c: In function ‘main’:
fork.c:5:1: warning: implicit declaration of function ‘fork’ [-Wimplicit-functio
n-declaration]
    5 | fork();
      | ^~~~
```

The code cannot be compiled because without #include <unistd.h> fork() cannot create a child process due to lack of access to the POSIX operating system API.

2.2 Differentiate between Parent and Child Process

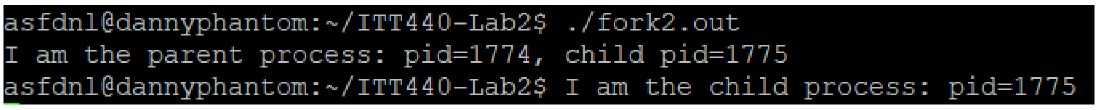
The fork() function will return:

- 0 if it is a child process
- Positive value if it is a parent
- -1 if there is an error

Please compile and run the following code on your Ubuntu/Linux

```
#include <stdlib.h>     /* needed to define exit() */

#include <unistd.h>     /* needed for fork() and getpid() */

#include <stdio.h>      /* needed for printf() */


Int main(int argc, char **argv) {

        int pid;        /* process ID */


        switch (pid = fork()) {

        case 0:                 /* a fork returns 0 to the child */

                printf("I am the child process: pid=%d\n", getpid());

                break;


        default:        /* a fork returns a pid to the parent */

                printf("I am the parent process: pid=%d, child pid=%d\n", getpid(), pid);

                break;


        case -1:        /* something went wrong */

                perror("fork");

                exit(1);

        }

        exit(0);

}
```

What is the Parent and Child Process PID? Please provide screenshot. **(0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ./fork2.out
I am the parent process: pid=1774, child pid=1775
asfdnl@dannyphantom:~/ITT440-Lab2$ I am the child process: pid=1775
```

Parent process ID is 1774 and child process ID is 1755.

What is the function or purpose of peror() function.**(0.5 Marks)**

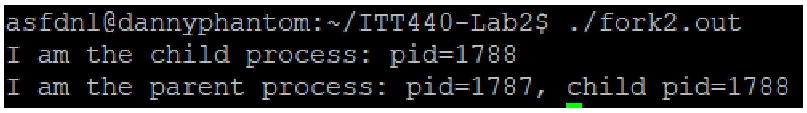
To return error message if something went wrong in creating the child process.

Now add the following to your code
#include <sys/wait.h>
wait(NULL);

Like below code, run and compile your code

```
#include <stdlib.h>     /* needed to define exit() */
#include <unistd.h>     /* needed for fork() and getpid()  */
#include <stdio.h>      /* needed for printf() */
#include <sys/wait.h>

Int main(int argc, char **argv) {
        int pid;        /* process ID */

        switch (pid = fork()) {
        case 0:                 /* a fork returns 0 to the child */
                printf("I am the child process: pid=%d\n", getpid());
                break;

        default:        /* a fork returns a pid to the parent */
                wait(NULL);
                printf("I am the parent process: pid=%d, child pid=%d\n", getpid(), pid);
                break;

        case -1:        /* something went wrong */
                perror("fork");
                exit(1);
        }
        exit(0);
}
```

What is the difference from previous code output, please also provide screenshot of the output, why is that? (**0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ./fork2.out
I am the child process: pid=1788
I am the parent process: pid=1787, child pid=1788
```

<u>The difference is the child process is executed first the the parent process. The child process finish its process first followed by the parent process.</u>

What is the function or purpose of wait() function.**(0.5 Marks)**

<u>wait(NULL) function in the parent process makes the parent process waits for the child process to finish first before executing.</u>

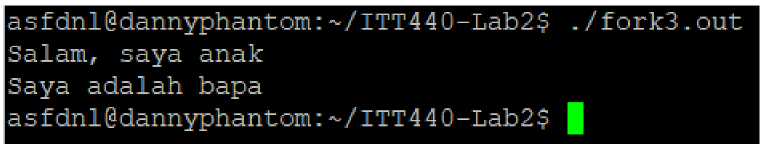
2.3 Using Function and Looping in Multi-process Program

To make things easier and modular, we often use function to execute a child or parent task.

Please compile and run the following code on your Ubuntu/Linux

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void childTask() {
  printf("Salam, saya anak tau\n");
}

void parentTask() {
  printf("Dan saya adalah bapaknya n");
}

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

  if(pid == 0) {
    childTask();
    exit(EXIT_SUCCESS);
  }
  else if(pid > 0) {
    wait(NULL);
    parentTask();
  }
  else {
    printf("Unable to create child process.");
  }

  return EXIT_SUCCESS;
}
```

How many times the program print its output? Please provide screenshot. Please briefly explain why it is easier to use function. **(0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ./fork3.out
Salam, saya anak
Saya adalah bapa
asfdnl@dannyphantom:~/ITT440-Lab2$
```

<u>The program prints its output twice. It is easier because user will just have to run the program once and the program will automatically create another process.</u>

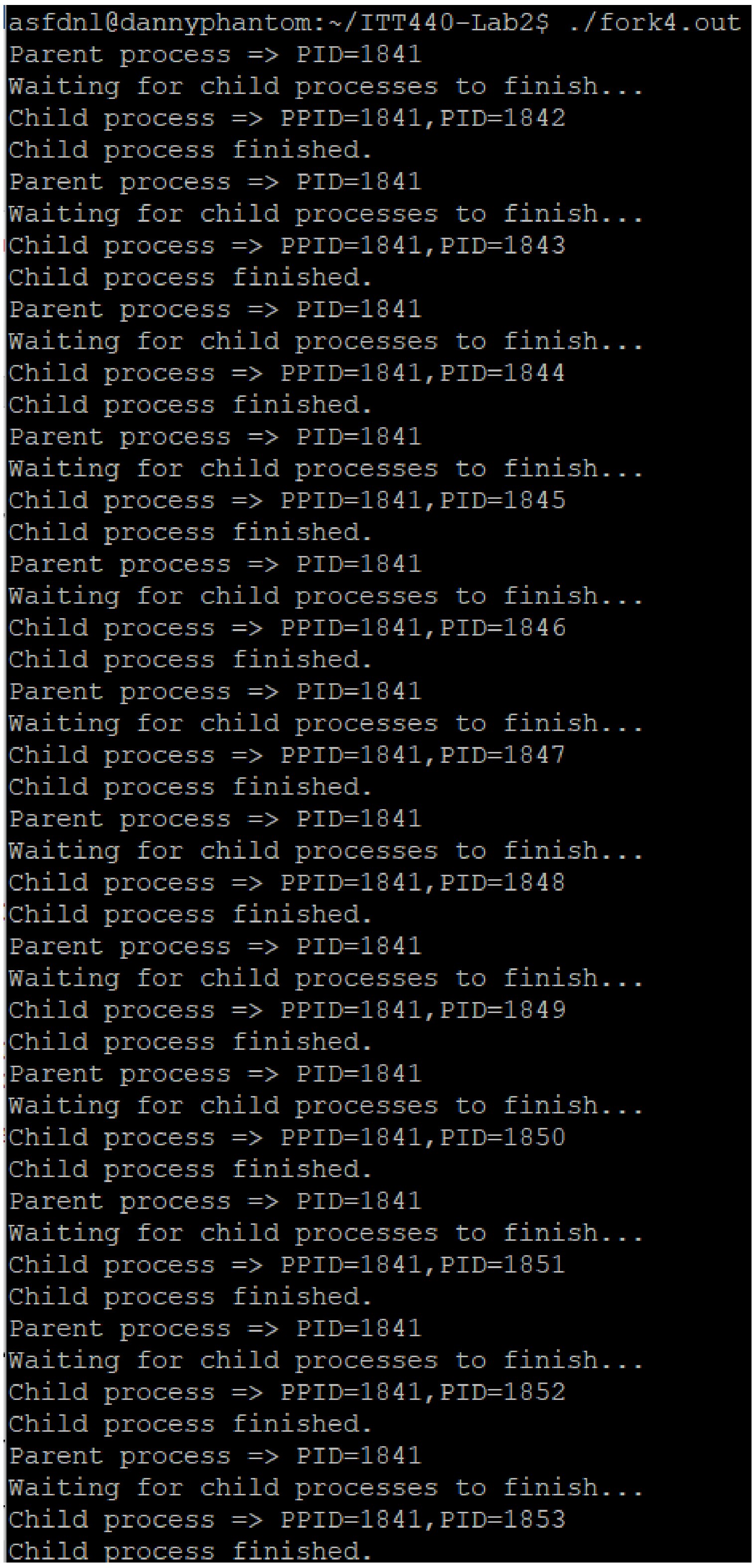
We can also use loop to create multiple child in our program.

Now, please compile and run the following code on your Ubuntu/Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void) {
  for(int i = 1; i < 13; i++) {
    pid_t pid = fork();

    if(pid == 0) {
      printf("Child process => PPID=%d,
PID=%d\n", getppid(), getpid());
      exit(0);
    }
    else  {
      printf("Parent process => PID=%d\n", getpid());
      printf("Waiting for child processes to finish...\n");
      wait(NULL);
      printf("child process finished.\n");
    }
  }

  return EXIT_SUCCESS;
}
```

How many child processes has been created? Please provide screenshot. **(0.5 Marks)**

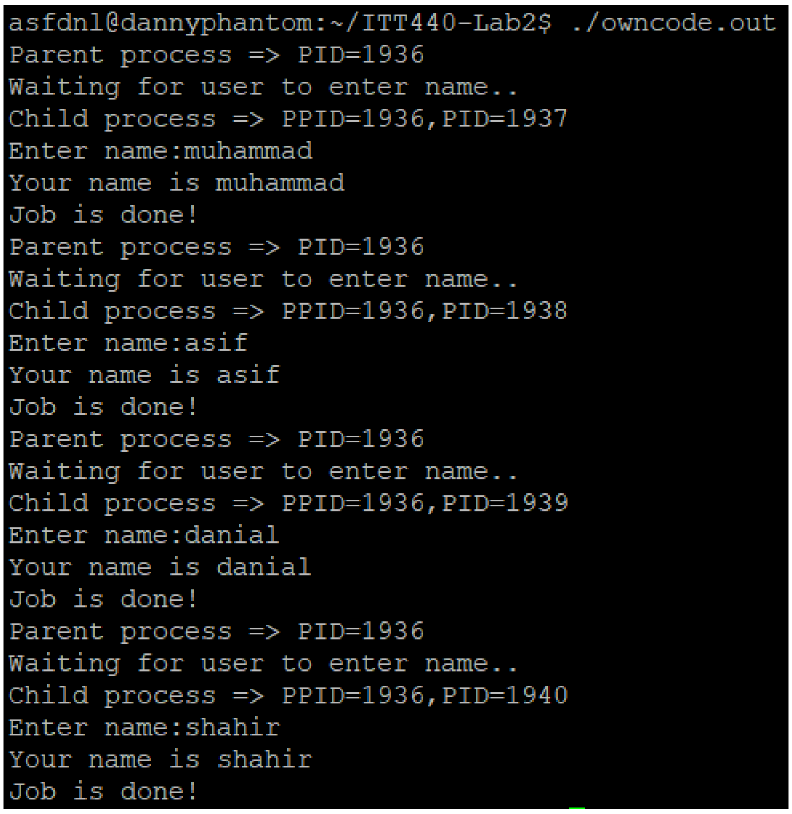
13 child processes.



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ./fork4.out
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1842
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1843
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1844
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1845
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1846
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1847
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1848
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1849
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1850
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1851
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1852
Child process finished.
Parent process => PID=1841
Waiting for child processes to finish...
Child process => PPID=1841,PID=1853
Child process finished.
```

2.4 Modify or create your own code

Now based on what you have learn in this lab, modify above code or create a new code to create a program which the 4-child process will ask user to enter its name and display back name which has been given. The parent process will then for all child to finish its job and then print out "Job is done" and then exit. **Your code must use fork, wait, function and loop to achieve the output**. Please provide screenshot of your output. **(0.5 Marks)**



```
asfdnl@dannyphantom:~/ITT440-Lab2$ ./owncode.out
Parent process => PID=1936
Waiting for user to enter name..
Child process => PPID=1936,PID=1937
Enter name:muhammad
Your name is muhammad
Job is done!
Parent process => PID=1936
Waiting for user to enter name..
Child process => PPID=1936,PID=1938
Enter name:asif
Your name is asif
Job is done!
Parent process => PID=1936
Waiting for user to enter name..
Child process => PPID=1936,PID=1939
Enter name:danial
Your name is danial
Job is done!
Parent process => PID=1936
Waiting for user to enter name..
Child process => PPID=1936,PID=1940
Enter name:shahir
Your name is shahir
Job is done!
```