**Lab 2: Fork System Call in C**

Student Name: MUHAMMAD SAFIY HAIKAL BIN MOHD FAUZI

Github Repository for Lab 2:

Matric No: 2021887274

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>

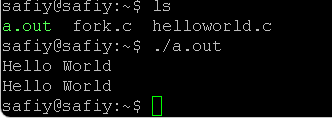
#include <unistd.h>

int main(){ fork(); printf("Hello World \n"); return 0;

}

How many times the program print its output? Please provide screenshot. Please briefly explain why.

**-2 hello world**



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.

-Error

Text

Description automatically generated

## 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.

Text

Description automatically generated

What is the function or purpose of peror() function.

-If the negative number return,it will exit the program.

Now add the following to your code

#include <sys/wait.h> wait(NULL);

Like below code, run and compile your code (Switch)

#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>

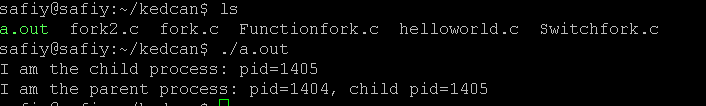
Int main(int argc, char \*\*argv) {

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?



What is the function or purpose of wait() function.

**- Used to make a parent process wait until one of its child processes finishes executing.**

## 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

#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.

Text

Description automatically generated

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");

}

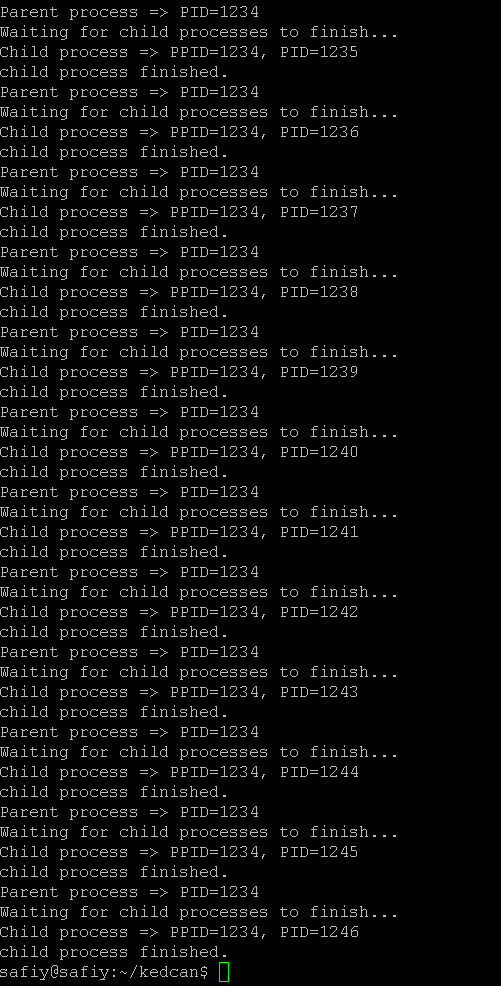
}

return EXIT\_SUCCESS;

}

How many child processes has been created? Please provide screenshot.

**-13 child processes**



## 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 wait 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.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid[4]; // array to store child process IDs

for (int i = 0; i < 4; i++) {

pid[i] = fork(); // create child processes

if (pid[i] == 0) { // child process

char name[50];

printf("Child %d: Enter your name: ", i+1);

fgets(name, sizeof(name), stdin);

printf("Child %d: Hello, %s", i+1, name);

exit(0);

} else if (pid[i] < 0) { // fork failed

printf("Error: Fork failed\n");

exit(1);

}

}

for (int i = 0; i < 4; i++) {

waitpid(pid[i], NULL, 0); // wait for each child process to finish

}

printf("Job is done\n");

return 0;

}