

Assignment 4

Samantha Licea Dominguez | z122301

Exercise 1

- Explain how to call **fork()** and how to work the process on your system.

The fork system call is used to create a new process called the **child process**. The process which does the **fork()** call and creates the new process is called the **parent process**. Both the parent and the child processes are executed at the same time but they reside on different memory spaces that have the same content and whatever operation is performed by one process won't affect the other. When the child process is created, both the parent and the child share the same PC.

There are no arguments in fork() function, and the return type is an integer. The following header files should be included:

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

Header	Usage
<sys/types.h>	Used for type pid_t for processes ID.
<unistd.h>	Definition of fork() .

The way of working the process on my system can be observed as follows:

Example 1

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

int main(){
    fork();
    printf("Using fork() system call \n");
    return 0;
}
```

In this example the string contained in the ***printf*** function is printed twice, one for the parent and one for the child.

Example 2

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

int main(){
    pid_t p;
    p = fork();
    if(p == -1){
        printf("There is an error while calling fork()\n");
    }
    if(p == 0){
        printf("We are in child process\n");
    }
    else{
        printf("We are in the parent process\n");
    }

    return 0;
}
```

In this example, the variable ***p*** stores the return value of ***fork()*** and then is compared to see if it was successful. We note now that the parent process is run first and then the child process is executed.

Excercise 2

- Add ***getpid()*** API to obtain the parent process's PID and print it in the ***printf()*** result like (this is parent (pid=XXXX)).
- Compare the parent ID, child ID. Is it the same?

Getpid() is the function used to get the process ID of the process that calls the function. The PID for the initial process is 1, and then each new process is assigned a new ID. Two types of IDs are present here: one is the current ID of the process PID and the other is the ID of the parent process PPID. For this, the header to be defined at the beginning of the program is:

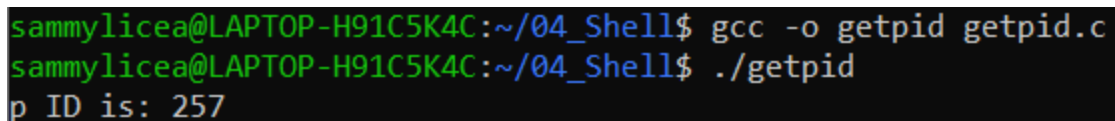
```
#include <unistd.h>
```

When some process is formed and running, a unique ID is assigned to it (process ID). The function **getpid()** returns the ID of the process that is currently called.

Example 1

```
#include <stdio.h>
#include <unistd.h>

int main(){
    int pid_t = getpid();
    printf("p ID is: %d\n",pid_t);
    return 0;
}
```



```
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ gcc -o getpid getpid.c
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ ./getpid
p ID is: 257
```

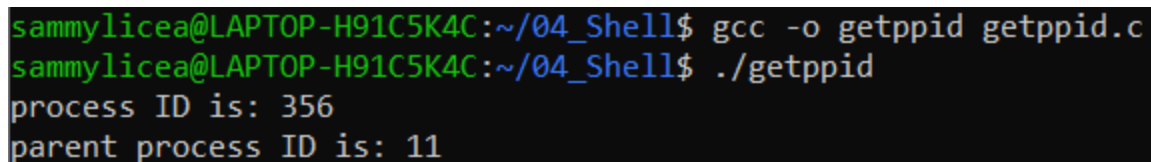
The above code and image shows the process ID of the function.

Example 2

```
#include <stdio.h>
#include <unistd.h>

int main(){
    int pid_a = getpid();
    int ppid_x = getppid();

    printf("process ID is: %d\n",pid_a);
    printf("parent process ID is: %d\n",ppid_x);
    return 0;
}
```



```
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ gcc -o getppid getppid.c
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ ./getppid
process ID is: 356
parent process ID is: 11
```

The output shows that the process ID is displayed first and then the parent's ID.

Example 3

```

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

int main(){
    for(int i = 1; i <= 3; i++){
        pid_t pid = fork();
        if(pid == 0){
            printf("This is a child process => PPID = %d, PID = %d\n", getppid(), getpid());
            exit(0);
        } else{
            printf("This is a parent process => PID = %d\n", getpid());
            printf("Wait until the child process is finished...\n");
            wait(NULL);
            printf("The child process is now finished.\n");
        }
    }

    return 0;
}

```

```

sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ gcc -o pids pids.c
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ ./pids
This is a parent process => PID = 455
Wait until the child process is finished...
This is a child process => PPID = 455, PID = 456
The child process is now finished.
This is a parent process => PID = 455
Wait until the child process is finished...
This is a child process => PPID = 455, PID = 457
The child process is now finished.
This is a parent process => PID = 455
Wait until the child process is finished...
This is a child process => PPID = 455, PID = 458
The child process is now finished.

```

The program above shows both the parent and the child processes IDs. We can see that the parent ID is the same in each case, but the child's ID changes.

Exercise 3

- Develop a command that creates processes, with the following requirements:

a. Modify the program to accept argument numbers (number of processes).

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

int main(int argc, const char *argv[]){
    if(argc != 2){
        printf("Input error: Too few arguments\n");
        return -1;
    }

    int lim = atoi(argv[1]);

    for(int i = 1; i <= lim; i++){
        pid_t pid = fork();

        if(pid == 0){
            printf("This is child process %d => PPID = %d, PID = %d\n", i, getppid(), getpid());
            exit(0);
        }else{
            wait(NULL);
        }
    }

    return 0;
}
```

```
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ gcc -o fork fork.c
sammylicea@LAPTOP-H91C5K4C:~/04_Shell$ ./fork 10
This is child process 1 => PPID = 280, PID = 281
This is child process 2 => PPID = 280, PID = 282
This is child process 3 => PPID = 280, PID = 283
This is child process 4 => PPID = 280, PID = 284
This is child process 5 => PPID = 280, PID = 285
This is child process 6 => PPID = 280, PID = 286
This is child process 7 => PPID = 280, PID = 287
This is child process 8 => PPID = 280, PID = 288
This is child process 9 => PPID = 280, PID = 289
This is child process 10 => PPID = 280, PID = 290
```

Excercise 4

- Compare the parent's process ID and child's ID. What kind of difference you can understand?

Taking into consideration all the examples and exercises that I have performed by now, I can tell that each time I run a command, the parent ID stays the same. If I only have one `fork()` command, the parent and the child will have different IDs. However if I have more than one `fork()` command, the ID of the parent stays the same in each case and the child created changes ID. So in the same execution, the parent ID will remain the same, while the child processes created on that execution will all be different.

Exercise 5

- Measure the time execution that creates 1, 100 and 1000 processes.
- Write your analysis on why such a result was obtained.

# of processes	real time	user time	sys time
1	0m0,307s	0m0,001s	0m0,306s
100	0m0,347s	0m0,023s	0m0,326s
1000	0m0,177s	0m0,127s	0m0,065s
10,000	0m1,718s	0m1,263s	0m0,602s

Here it can be seen that as the number of processes increase, the time of execution increases as well. We can observe that the gap of real time between 1 and 10,000 processes is of 1,411s (5.6 times bigger), the user time is of 1,262s (1,263 times bigger), and the system time is of 0,296s (1.97 times bigger). So, it can be concluded that the number of processes is directly proportional to the amount spent on executing the programs.

Exercise 6

- What kind of application can be considered with the many processes with one parent?

Maybe when trying to search for a register in a large data base. The child processes can be assigned to look for the values of the keys proportioned. Another idea could be when trying to establish P2P or client-server connections where one server has the parent process and the connections are the child processes.

References

Fork System Call Linux

The fork system call is used to create a new processes. The newly created process is the child process. The process which calls fork and creates a new process is the parent process. The child and

😊 <https://linuxhint.com/fork-system-call-linux/>

[illegible]

Calling getpid function in C with Examples

Getpid() is the function used to get the process ID of the process that calls that function. The PID for the initial process is 1, and then each new process is assigned a new Id. It is a simple approach to

😊 <https://linuxhint.com/getpid-function-c-examples/>

```
code1.c
Open Save

#include <stdio.h>

int main(void)
{
    //getting process id and storing in a variable
    int pid_a = getpid();
    //getting parent function process id and storing in a variable
    int ppid_x = getppid();

    //printing the ids of the processes
    printf("process id is : %d\n", pid_a);
    printf("parent process id is : %d\n", ppid_x);
    return 0;
}
```