Bharati Vidyapeeth (Deemed To University)
College of Engineering, Pune

DEPARTMENT OF COMPUTER ENGINEERING

OPERATING SYSTEM

EXPERIMENT NO. - 02

AIM- Demonstrate the process creation.

NAME: Kushagra Agarwal

PRN NO.: 1814110037

ROLL NO.: 47

CLASS: B.Tech Sem-VI Div-1

Theory-

Process creation is achieved through the fork() system call. The newly created process is called the child process and the process that initiated it (or the process when execution is started) is called the parent process. After the fork() system call, now we have two processes - parent and child processes. How to differentiate them? Very simple, it is through their return values.

A process can terminate in either of the two ways –
Abnormally, occurs on delivery of certain signals, say terminate signal.
Normally, using _exit() system call (or _Exit() system call) or exit() library function.

The difference between _exit() and exit() is mainly the cleanup activity. The exit() does some cleanup before returning the control back to the kernel, while the _exit() (or _Exit()) would return the control back to the kernel immediately.

System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork():

If fork() returns a negative value, the creation of a child process was unsuccessful. fork() returns a zero to the newly created child process. fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process. Therefore, after the system call to fork(), a simple test can tell which process is the child. Please note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

Parent Process

All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the parent process. In other words, a parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.

On the success of a fork() system call, the PID of the child process is returned to the parent process and 0 is returned to the child process. On the failure of a fork() system call, -1 is returned to the parent process and a child process is not created.

Child Process

A child process is a process created by a parent process in operating system using a fork() system call. A child process may also be called a subprocess or a subtask.

A child process is created as its parent process's copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel.
If a child process exits or is interrupted, then a SIGCHLD signal is send to the parent process.

Fork System Call

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 parent processes are executed concurrently.
But the child and parent processes reside on different memory spaces. These memory spaces have same content and whatever operation is performed by one process will not affect the other process.
When the child processes is created; now both the processes will have the same Program Counter (PC), so both of these processes will point to the same next instruction. The files opened by the parent process will be the same for child process.
The child process is exactly the same as its parent but there is difference in the processes ID's:

1. The process ID of the child process is a unique process ID which is different from the ID's of all other existing processes.
2. The Parent process ID will be the same as that of the process ID of child's parent.

Properties of Child Process

The following are some of the properties that a child process holds:

1. The CPU counters and the resource utilizations are initialized to reset to zero.
2. When the parent process is terminated, child processes do not receive any signal because PR_SET_PDEATHSIG attribute in prctl() is reset.
3. The thread used to call fork() creates the child process. So the address of the child process will be the same as that of parent.
4. The file descriptor of parent process is inherited by the child process. For example the offset of the file or status of flags and the I/O attributes will be shared among the file descriptors of child and parent processes. So file descriptor of parent class will refer to same file descriptor of child class.
5. The open message queue descriptors of parent process are inherited by the child process. For example if a file descriptor contains a message in parent process the same message will be present in the corresponding file descriptor of child process. So we can say that the flag values of these file descriptors are same.
6. Similarly open directory streams will be inherited by the child processes.
7. The default Timer slack value of the child class is same as the current timer slack value of parent class.

Name- Kushagra Agarwal
PRN- 1814110037
Roll No.-47

Properties that are not inherited by Child process

The following are some of the properties that are not inherited by a child process:

1. Memory locks
2. The pending signal of a child class is empty.
3. Process associated record locks (fcntl())
4. Asynchronous I/O operations and I/O contents.
5. Directory change notifications.
6. Timers such as alarm(), setitimer() are not inherited by the child class.

fork() in C

There are no arguments in fork() and the return type of fork() is integer. You have to include the following header files when fork() is used:

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

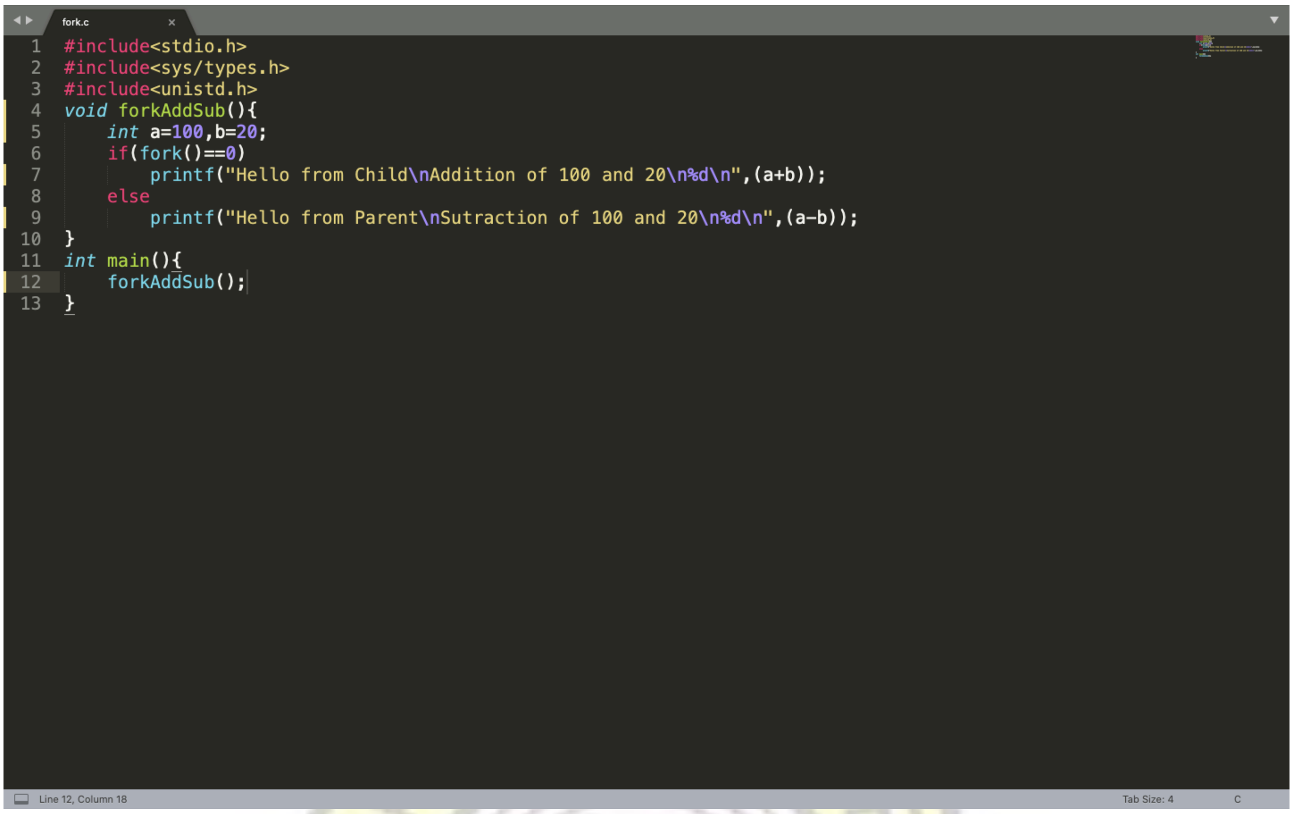
When working with fork(), <sys/types.h> can be used for type ***pid_t*** for processes ID's as pid_t is defined in <sys/types.h>.
The header file <unistd.h> is where fork() is defined so you have to include it to your program to use fork().
The return type is defined in <sys/types.h> and fork() call is defined in <unistd.h>. Therefore, you need to include both in your program to use fork() system call.

Example: Calling fork()

Consider the following example in which we have used the fork() system call to create a new child process:

Name- Kushagra Agarwal
PRN- 1814110037
Roll No.-47

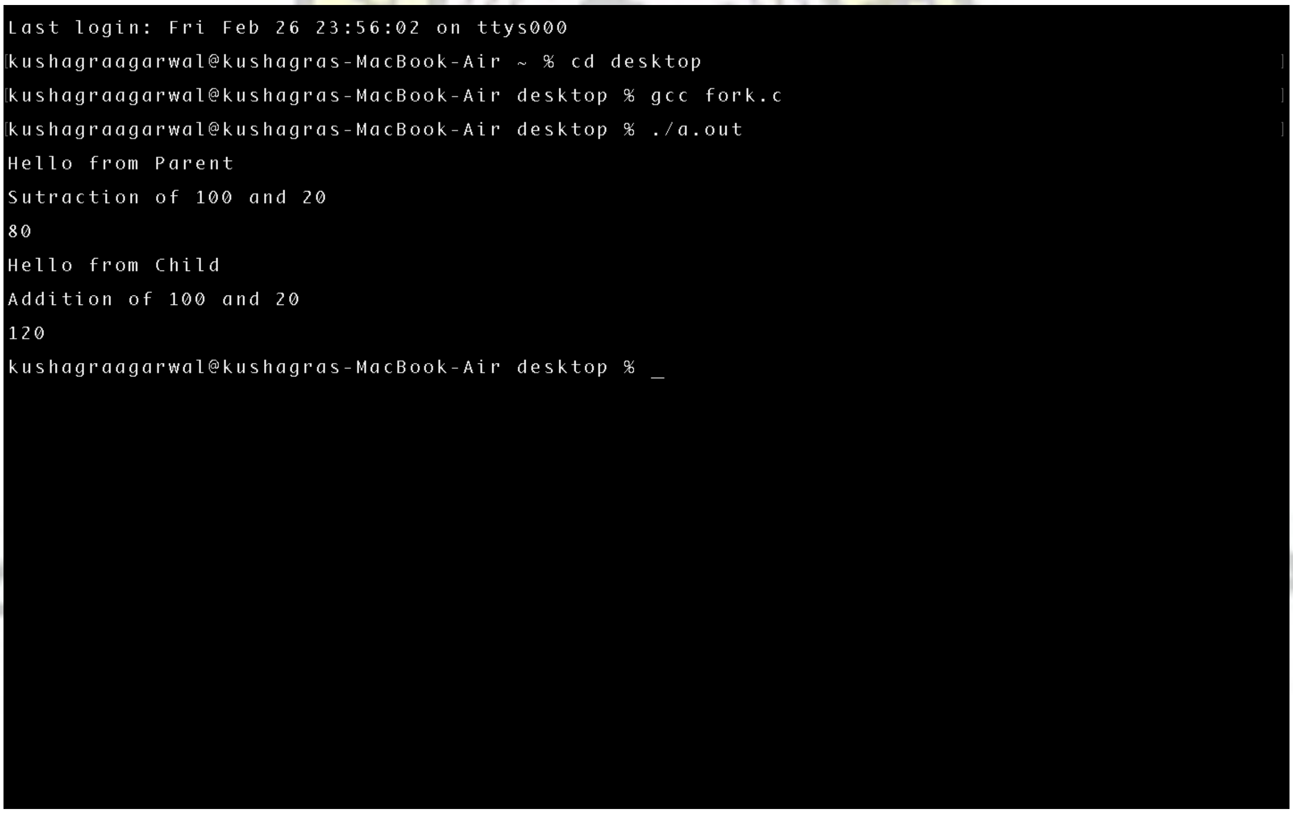
CODE:



fork.c

```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
void forkAddSub(){
    int a=100,b=20;
    if(fork()==0)
        printf("Hello from Child\nAddition of 100 and 20\n%d\n",(a+b));
    else
        printf("Hello from Parent\nSutraction of 100 and 20\n%d\n",(a-b));
}
int main(){
    forkAddSub();
}
```

Line 12, Column 18 Tab Size: 4 C



```
Last login: Fri Feb 26 23:56:02 on ttys000
kushagraagarwal@kushagras-MacBook-Air ~ % cd desktop
kushagraagarwal@kushagras-MacBook-Air desktop % gcc fork.c
kushagraagarwal@kushagras-MacBook-Air desktop % ./a.out
Hello from Parent
Sutraction of 100 and 20
80
Hello from Child
Addition of 100 and 20
120
kushagraagarwal@kushagras-MacBook-Air desktop % _
```

Conclusion-

In this practical, we learned about process creation.