Practical -5 Process creation

PS Command

To show the processes for the current running shell run ps. If nothing else is running this will return information on the shell process being run and the ps command that is being run.

```
ps
PID  TTY          TIME CMD
5763 pts/3    00:00:00 sh
8534 pts/3    00:00:00 ps
```

The result contains four columns of information.

- PID - the number of the process
- TTY - the name of the console that the user is logged into
- TIME- the amount of CPU in minutes and seconds that the process has been running
- CMD - the name of the command that launched the process

Sleep

The only purpose of sleep command is to block or delay the execution of a particular script for a defined amount of time. This is a useful mechanism where you have to wait for a specific operation to complete before the start of other activity in your shell scripts.

getppid() and getpid()

1. **getppid() :** returns the process ID of the parent of the calling process. If the calling process was created by the **fork()** function and the parent process still exists at the time of the getppid function call, **this function returns the process ID of the parent process.**
2. **getpid() :** returns the process ID of the process
3. **getuid**() returns the real user ID of the calling process.
4. **getgid**() returns the real group ID of the calling process.

Program 1

```
//This program displays various IDs related to a process

#include<stdio.h>

int main()
{
        system("ps");
        sleep(2);
        printf("PID=%d, PPID=%d\n",getpid(),getppid());
        printf("UID=%d, GID=%d\n",getuid(),getgid());
        exit(0);
}
```

Compile Program : gcc -o myprog myprog.c
Run Program: ./myprogram

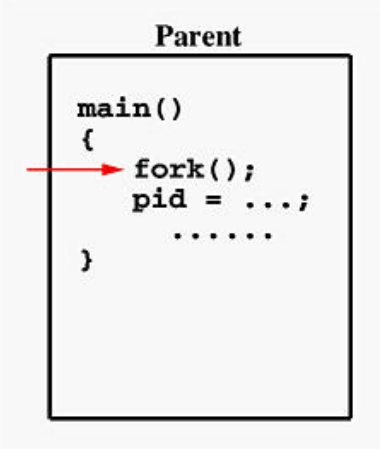
Program 2

Process States:

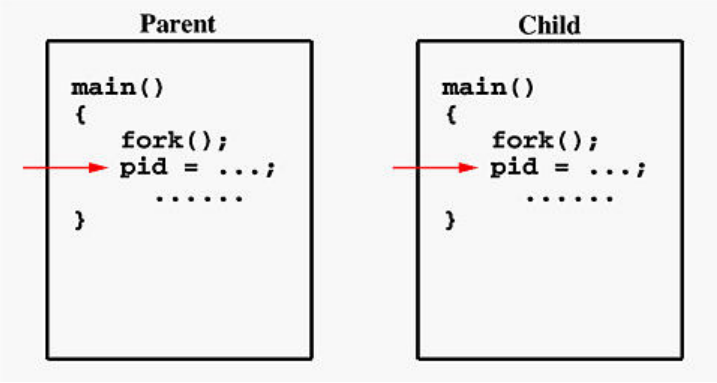
- Created: first stage
- Ready-to-Run (or Run able): process is ready but does not have CPU.
- Running: running in either user mode/ kernel mode
- Sleeping: not doing work e.g. I/O

System call fork():

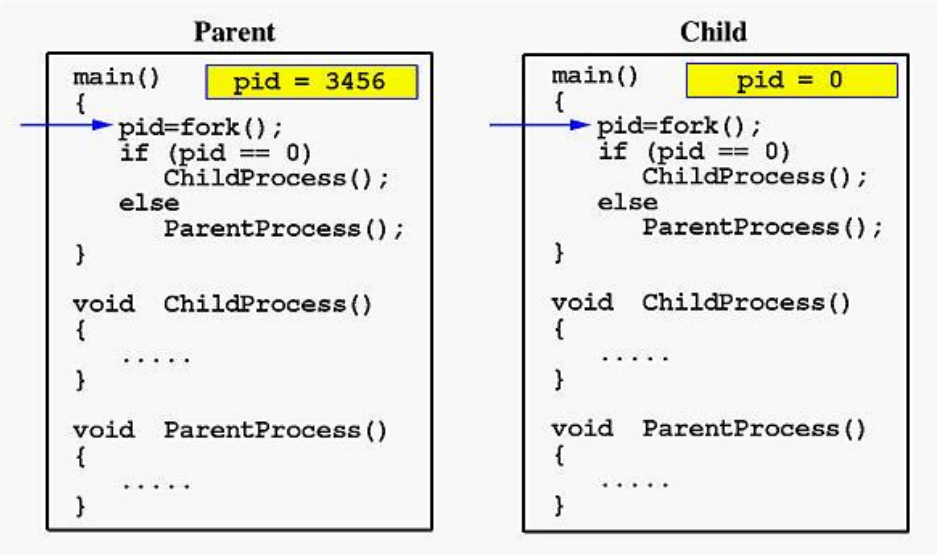
- **Is used to create processes. It takes no arguments and returns a process Id**.
- The purpose of fork() is to create a new process, which becomes the child process of the caller.
- After a new child process is created, **both processes will execute the next instruction following** the fork() system call.
- If fork() returns a negative value, if 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 an integer. Moreover, a process can use function getpid() to retrieve the process ID.



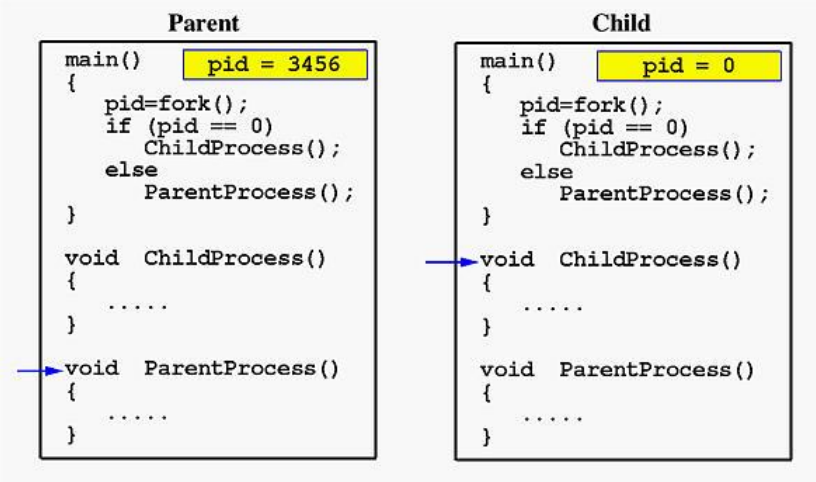
if the call to fork() is executed successfully, Lnix will
make two identical copies of address spaces, one for the parent and the other for the child. Both processes will start their execution at the next statement following the fork() call.



E.g



since pid is non-zero, it calls function ParentProcess(). On the other hand, the child has a zero pid and calls ChildProcess()




```
//This program shows simple fork operation to create new process

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

int main()
{
        pid_t pid;
        printf("\nBefore fork");

        pid=fork();
        if(pid>0)
        {
                sleep(1);
                printf("Parent process\n");
                printf("PID=%d,PPID=%d,child PID=%d\n",getpid(),getppid(),pid);

        }
        else if(pid==0)
        {
                printf("Child process\n");
                printf("PID=%d,PPID=%d\n",getpid(),getppid());
        }
        else
        {
                printf("Fork error\n");
                exit(1);
        }
        printf("Both process continues form here...\n");
        exit(0);
}
```

Program 3

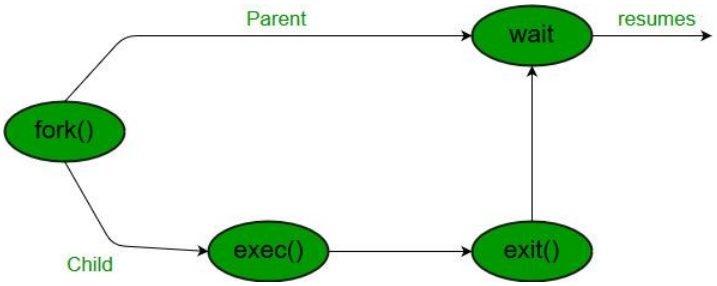
```
//This program changes child's environment variables and check the effect

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

int main()
{
        pid_t pid;
        int x=100;

        printf("Before fork...\n");
        pid=fork();
        switch(pid)
        {
                case -1: printf("Fork error...\n");
                         exit(1);

                default:
                        sleep(2);
                        printf("Parent process...\n");
                        printf("value of x=%d \n",x);
                        exit(0);
                case 0:
                        printf("Child process...\n");
                        printf("initial value of x=%d \n",x);
                        x=200;
                        printf("New value of x=%d  \n",x);
                        break;
                        exit(0);
        }
}
```

Wait() System Call

The parent process may then issue a wait system call, which suspends the execution of the parent process while the child executes. When the child process terminates, it returns an

exit status to the operating system, which is then returned to the waiting parent process. The parent process then resumes execution.

WEXITSTATUS(status) : returns the exit status of the child.



Program 4

```
//This program shows use of wait() system call

#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<fcntl.h>

int main()
{
        int exitval=10,exitstatus;
        switch(fork())
       {
       case 0:
                printf("CHILD: Terminating with exit value %d\n",exitval);
                exit(exitval);
       default:
                wait(&exitstatus);
                printf("PARENT: Child terminated with value %d\n",
                WEXITSTATUS(exitstatus));
                exit(20);
     }
}
```

Exercise

1.	Write a program having two value a=10 and b=10. Child process will change to value of a and b to 20 and 30 respectively. Check the value of a and b in parent process.
2	Write a program having two value x=5 and y=3. Parent process print the multiplication of x and y . child process print the addition of x and y.
3	Write a program which print process id and parent process id of processes. In which Parent process should finish first then child process