Practical 2

Process

Process : A process is a program in a state of execution.

Process Attributes :

- PID, PPID : Process ID and parent process ID
- PID – process id : Unique number assigned for each process in increasing order when they are created
- PPID – parent PID : The PID of the parent from which it was cloned. Linux uses fork-and-exec model to create new process
- The init daemon has a PID of 1 and a PPID of 0
- UID, EUID :User ID and Effective user ID
- GID, EGID :Group ID and Effective group ID

- States of the Process : Running ,Sleep, Ready
  - [Read-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
  - Zombie: when child dies, address space de allocated and considered as zombie.]
- Kinds of Process: Parent, Child, Orphan, Daemon, Zombie.

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 zsh
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

$ ps

```
    PID TTY          TIME CMD
    1074 pts/2    00:00:00 bash
    1280 pts/2    00:00:00 parentTest.exe
    1281 pts/2    00:00:00 childTest.exe <defunct>                [abnormal (zombie)]
    1288 pts/2    00:00:00 ps
```

$ ps -l

```
Warning: /boot/System.map has an incorrect kernel version.
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
000 S   561  1074  1073  0  76   0   -   628 11a418 pts/2    00:00:00 bash
000 S   561  1301  1074  0  70   0   -   436 11f22b pts/2    00:00:00 parentTes
004 Z   561  1302  1301  0  70   0   -     0 119ffb pts/2    00:00:00 childTest
000 R   561  1320  1074  0  77   0   -   646      - pts/2    00:00:00 ps
```

Field	Contents
USER	Username of the process's owner
PID	Process ID
%CPU	Percentage of the CPU this process is using
%MEM	Percentage of real memory this process is using
VSZ	Virtual size of the process, in kilobytes
RSS	Resident set size (number of 1K pages in memory)
TT	Control terminal ID
STAT	Current process status: R = Runnable D = In disk (or short-term) wait I = Sleeping (> 20 sec) S = Sleeping (< 20 sec) T = Stopped Z = Zombie Additional Flags: > = Process has higher than normal priority N = Process has lower than normal priority < = Process is exceeding soft limit on memory use A = Process has requested random page replacement S = Process has asked for FIFO page replacement V = Process is suspended during a **vfork** E = Process is trying to **exit** L = Some pages are locked in core X = Process is being traced or debugged s = Process is a session leader (head of control terminal) W= Process is swapped out + = Process is in the foreground of its control terminal
STARTED	Time the process was started
TIME	CPU time the process has consumed
COMMAND	Command name and arguments[a]

Sleep

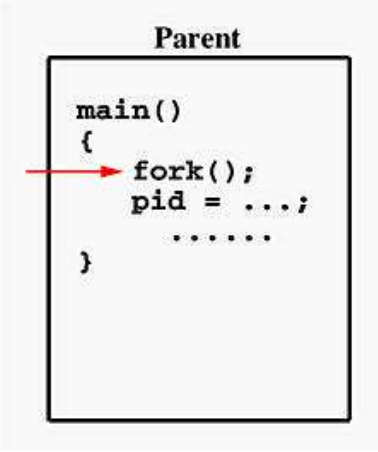
The only purpose of sleep command is to block or delay the execution of a particular script for a defined amount of time. This is a useful mechanism where you have to wait for a specific operation to complete before the start of other activity in your shell scripts.

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 calling process. This is often used by routines that generate unique temporary filenames.
3. **getuid**() returns the real user ID of the calling process.
4. **getgid**() returns the real group ID of the calling process.

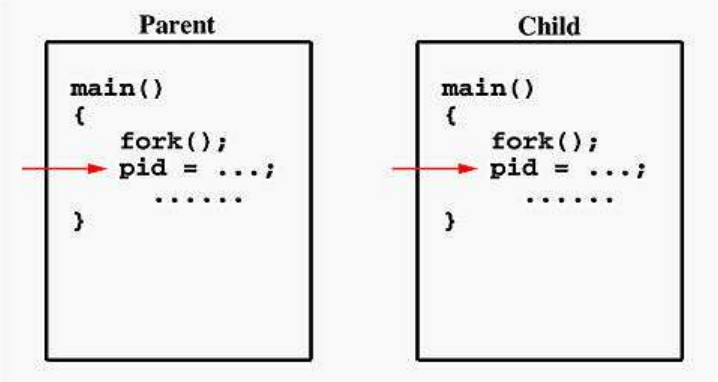
System call fork():

- 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 assigned to this process.

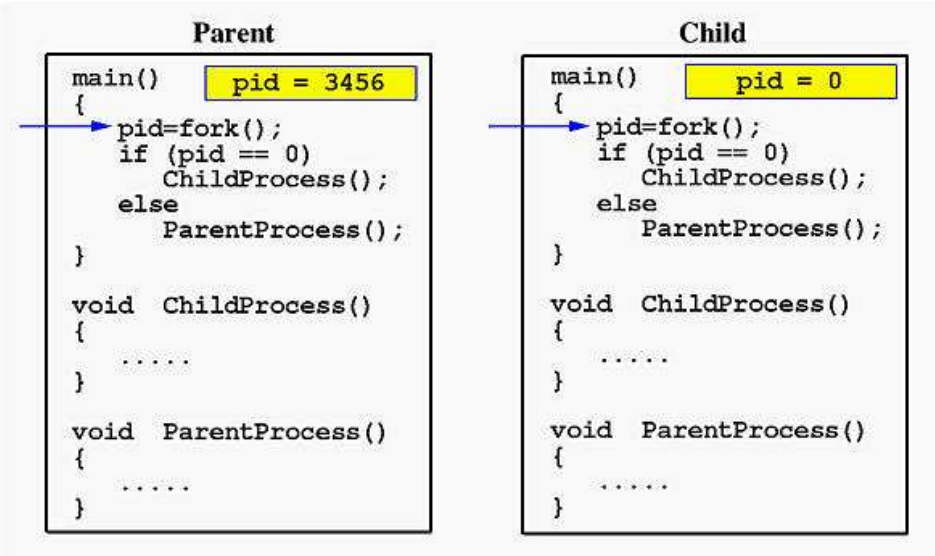


if the call to fork() is executed successfully, Linux will make two identical copies of address spaces, one for the parent and the other for the child.

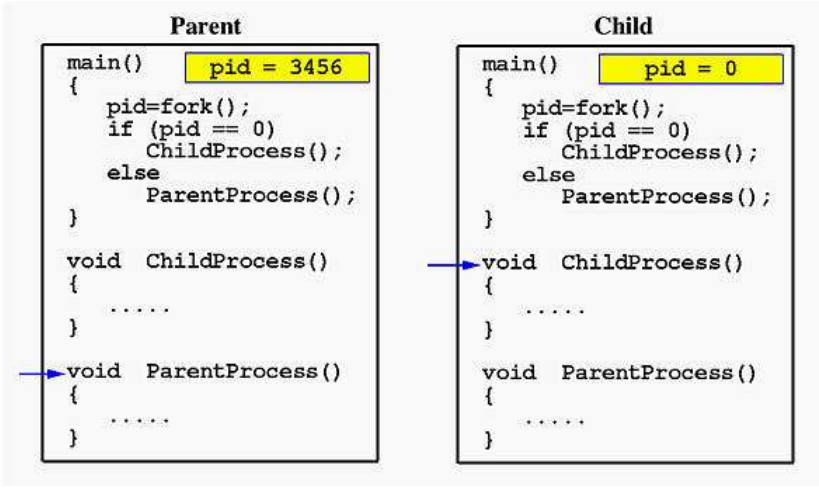
Both processes will start their execution at the next statement following the fork() call.



E.g



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



Program 1

```
// Running process in Background
#include <stdio.h>
void main()
{
        long k;
        for(k = 0; k <= 4000000; k++);
        printf("k is %d\n",k);
}
```

Compile using program using cc and run it as background
Running the program: ./a.out &
Compile Program : gcc -o myprog myprog.c
Run Program: ./myprogram

Program 2

```
// Process identification
#include<stdio.h>
int main()
{
        int pid;
        pid=getpid();
        printf("process ID=%d \n",pid);

}
```

Program 3

```
// Parent Process identification
#include<stdio.h>
int main()
{
        int ppid;
        ppid=getppid();
        printf("Parent process ID=%d \n",ppid);

}
```

Program 4

```
//This program displays various IDs related to a process
#include<stdio.h>
#include<stdlib.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);
}
```

Program 5

```
// Creation of Parent and child process
#include<stdio.h>
#include<stdlib.h>
int main()
{
        fork();
        printf("Hello world\n");

}
```

Program 6

```
//Creation of Parent and child process
#include<stdio.h>
#include<stdlib.h>
int main()
{
        printf(“this is to demonstrate the fork()\n”);
        fork();
        printf("Hello world\n");

}
```

Program 7

```
// Creation of Parent and child process
#include<stdio.h>
#include<stdlib.h>
int main()
{
        fork();
        fork();
        printf("Hello world\n");

}
```

Program 8

```
//Parent and child
#include<stdio.h>
#include<stdlib.h>
int main()
{
        fork();
        fork();
        fork();
        printf("Hello world\n");

}
```

Program 9

```
//Parent and child
#include<stdio.h>
#include<stdlib.h>
int main()
{
        fork();
        printf(“Hello\n”);
        fork();
        printf(“Hi\n”);
        fork();
        printf("Hello world\n");

}
```

Program 10

```
//Parent and child process id
#include<stdio.h>
#include<stdlib.h>
int main()
{
        fork();
        printf("Process id %d\n”,getpid());

}
```

Program 11

```
//Parent and child process id
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if(pid  > 0)
                printf("Parent Process id %d\n”,pid);

}
```

Program 12

```
// the pid of child process is always zero
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if(pid  == 0)
                printf("child process \n”);

}
```

Program 13

```
// the pid of child process if fork is successful else parent pid
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if(pid  == 0)
        {
                printf("I am the child, my process ID is %d \n”,getpid());
                printf("the child’s parent process ID is %d \n”,getppid());
        }
        else
        {
                printf("I am the parent, my process ID is %d \n”,getpid());
                printf("the parent’s parent process ID is %d \n”,getppid());
        }
}
```

Program 14

```
// the orphan process
#include<stdlib.h>
#include<stdio.h>
int main()
{
        int pid;
        pid = fork();
        if(pid  == 0)
        {
                printf("I am the child, my process ID is %d \n”,getpid());
                printf("the child’s parent process ID is %d \n”,getppid());
                sleep(20);
                printf("I am the child, my process ID is %d \n”,getpid());
                printf("the child’s parent process ID is %d \n”,getppid());

        }
        else
        {
                printf("I am the parent, my process ID is %d \n”,getpid());
                printf("the parent’s parent process ID is %d \n”,getppid());
        }
}
```

Program 15

```
// run this program in background and check the process table with ps –el command
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if(pid  == 0)
        {
                printf("I am the child, my process ID is %d \n”,getpid());
                printf("the child’s parent process ID is %d \n”,getppid());
                sleep(20);
                printf("I am the child, my process ID is %d \n”,getpid());
                printf("the child’s parent process ID is %d \n”,getppid());

        }
```

```
        else
        {
                sleep(10);
                printf("I am the parent, my process ID is %d \n”,getpid());
                printf("the parent’s parent process ID is %d \n”,getppid());
        }
}
```

Program 16

```
// Zombie process
#include<stdio.h>
#include<stdlib.h>

int main()
{
        fork();
        if(pid > 0)
        {
                printf("I am the parent \n”);
                sleep(50);
        }
}
```

Run this program in background by
./a.out &
The run the ps –el command

Program 17

```
//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 18

```
//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);
        }
}
```