Experiment No. 3

Title: Creating and Managing Child Processes in Linux Using `fork()`, `getpid()`, `getppid()`, `wait()`, and `waitpid()`

Aim: a. Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call.

b. Explore wait and waitpid before termination of process.

Theory:

a. Create a child process in Linux using the fork system call.

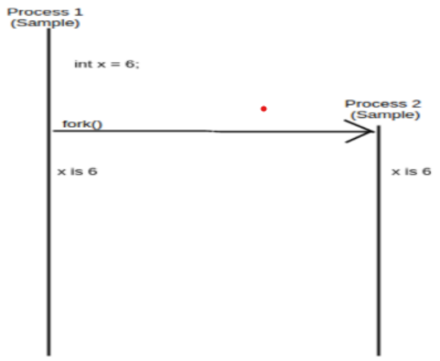
fork() System Call

A Process can create a new child process using fork() system call.This new child process created through fork() call will have same memory image as of parent process i.e. it will be duplicate of calling process but will have different process ID.
For example,

Suppose there is a Process "Sample" with Process ID 1256 and parent ID 12. Now as soon as this process calls the fork() function, a new process will be created with same memory image but with different process ID.

Also, process which has called this fork() function will become the parent process of this new process i.e.
Process 1: Sample (pid=1341 | Parent Process ID = 12)
After calling fork() system call,
Process 1: Sample (pid=1341 | Parent Process ID = 12)
Process 2: Sample (pid= 4567 | Parent Process ID = 1341)

As memory image of new child process will be the copy of parent process's memory image. So, all variables defined before fork() call will be available in child process with same values.



If fork() call is successful then code after this call will be executed in both the process. Therefore, fork() function's return value will be different in both the process's i.e.
If fork() call is successful then it will,

- Return 0 in child process.
- Return process id of new child process in parent process.

If fork() call is unsuccessful then it will return -1.

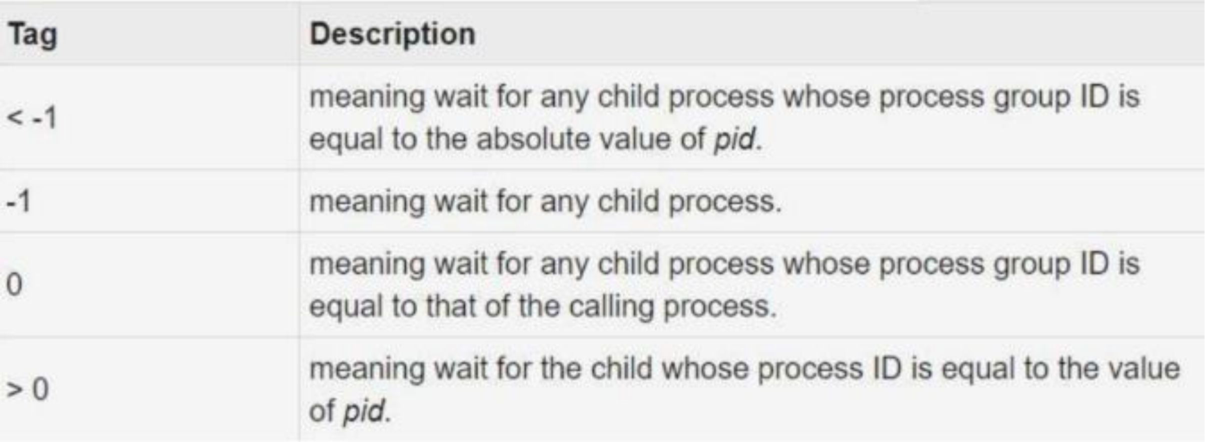
b. Explore wait and waitpid before termination of process

wait() and waitpid()

The wait() system call suspends execution of the current process until one of its children terminates. The call wait(&status) is equivalent to:

```
  waitpid(-1, &status, 0);
```

The waitpid() system call suspends execution of the current process until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children, but this behaviour is modifiable via the options argument, as described below.
The value of pid can be:



Tag	Description
< -1	meaning wait for any child process whose process group ID is equal to the absolute value of *pid*.
-1	meaning wait for any child process.
0	meaning wait for any child process whose process group ID is equal to that of the calling process.
> 0	meaning wait for the child whose process ID is equal to the value of *pid*.

Program:

a)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
// Driver code
int main()
{
int pid, pid1, pid2;
pid = fork();
if(pid == 0)
{
   sleep(3);
   print(“child[1] - - > pid = % = %d and ppid = %d \n”, getpid( ), getppid( ));
}
else {
   pid1 = fork( );
   if (pid1 == 0)
   {
      sleep(2);
      print(“child[2] - - > pid = % = %d and ppid = %d \n”, getpid( ), getppid( ));
    }
   else {
      pid2 = fork( );
      if (pid2 == 0)
     {
         sleep(2);
         print(“child[3] - - > pid = % = %d and ppid = %d \n”, getpid( ), getppid( ));
     }
     else
     {
         sleep(3);
          printf(“parent - - > pid = %d \n”,getpid( ) );
       }
    }
  }
 return 0;
}
```

b)

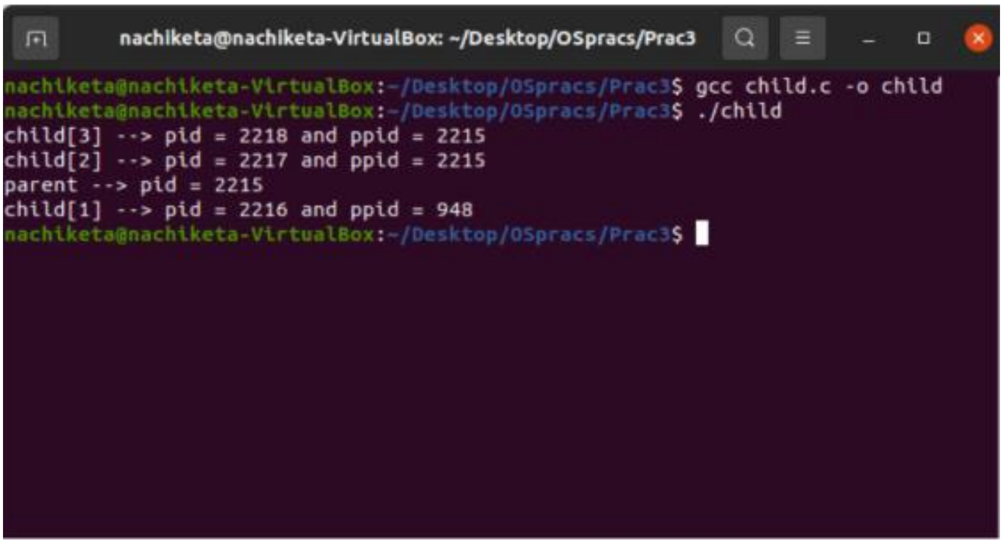
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
void waitexample()
{
 int i, stat;
pid_t
pid[5];
for (i=0; i<5; i++)
{
if ((pid[i] = fork()) == 0){
sleep(1);
exit(100 + i);
}
```

```
}
// Using waitpad( ) and printing exit status
// of children
for (i=0; i<5; i++)
{
pid_t cpid = waitpid(pid[i], &stat,0);
if (WIFEXITED(stat)){
printf("Child %d terminated with status: %d\n",cpid, WEXITSTATUS(stat));
}
//Driver code

int main( ){
waitexample( );
return 0;
}
```

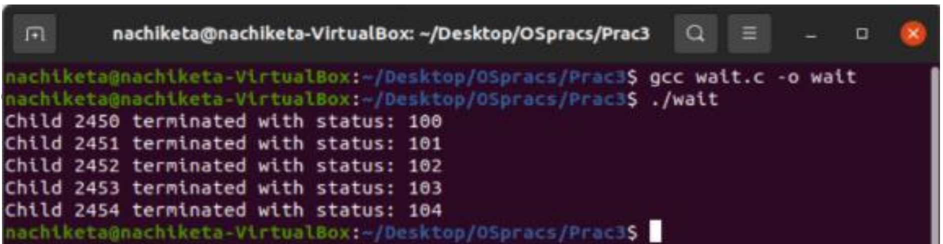
Output:

a)



```
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ gcc child.c -o child
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ ./child
child[3] --> pid = 2218 and ppid = 2215
child[2] --> pid = 2217 and ppid = 2215
parent --> pid = 2215
child[1] --> pid = 2216 and ppid = 948
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ 
```

b)



```
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ gcc wait.c -o wait
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ ./wait
Child 2450 terminated with status: 100
Child 2451 terminated with status: 101
Child 2452 terminated with status: 102
Child 2453 terminated with status: 103
Child 2454 terminated with status: 104
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac3$ 
```

Outcome:

Hence , Study of process system calls has been done.

Name: Khaire Chirag Gautam
Roll no.: 48. Batch: A2. Div: A