Experiment No 4
Class: SE Comp
Year: 2020-21
Performed by: Danyl Fernandes, 72

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:

- In OS, the fork() system call is used by a process to create another process.
- The process that used the fork() system call is the parent process & process consequently created is known as child process.

Process:

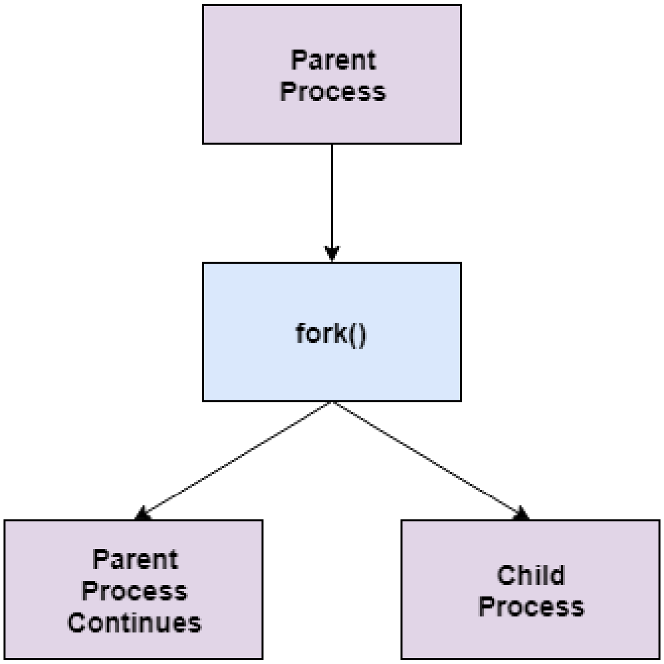
- A process is basically a program in execution. The execution of a process must progress in a sequential fashion.
- A process is defined as an entity which represents the basic unit of work to be implemented in the system.
- When a program is loaded into the memory and it becomes a process, it can be divided into four sections — stack, heap, text and data.
- A Process Control Block is a data structure maintained by the Operating System for every process.
- The PCB is identified by an integer process ID (PID).
- A PCB keeps all the information needed to keep track of a process

Parent Process:

- All the processes in the 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 has 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 an 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 sent to the parent process.
- A diagram that demonstrates parent and child process is given as follows –



System Calls:

- The interface between a process and an operating system is provided by system calls.
- In general, system calls are available as assembly language instructions.
- They are also included in the manuals used by the assembly level programmers.
- System calls are usually made when a process in user mode requires access to a resource.
- Then it requests the kernel to provide the resource via a system call.

Types of System Calls:
There are mainly five types of system calls. These are explained in detail as follows

- **Process Control**
    - These system calls deal with processes such as process creation, process termination etc
- **File Management**
    - These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
- **Device Management**
    - These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc
- **Information Maintenance**
    - These system calls handle information and its transfer between the operating system and the user program.
- **Communication**
    - These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Conclusion:

In this experiment, we understood the concept of system calls, obtaining process id using getpid, getppid.

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:

Code:

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

int main() {
int childpid,count1 = 0,count2 = 0;
childpid = fork();

if(childpid == 0)  {
     printf("This is a child process \n");

     while(count1 < 10) {
          printf("Child process : %d \n",count1);
          printf("Child process id : %d \n",getpid());
          printf("Parent process id : %d \n",getppid());
          sleep(1);
          count1++;
     }
} else {
     printf("This is a parent process\n");
     while(count2 < 20)  {
          printf("Parent process : %d \n",count2);
          printf("Parent process id :%d \n",getpid());
          sleep(1);
          Count2++;
     }
}

return 0;
}
```

Output:

```
dan@home:~/Desktop$ gedit p4_1.c
dan@home:~/Desktop$ gcc -o p4_1 p4_1.c
dan@home:~/Desktop$ ./p4_1
```

```
This is a parent process
Parent process : 0
Parent process id :3256

This is a child process
Child process : 0
Child process id : 3257

Parent process id : 3256
Parent process : 1
Parent process id :3256

Child process : 1
Child process id : 3257
Parent process id : 3256

Parent process : 2
Parent process id :3256

Child process : 2
Child process id : 3257
Parent process id : 3256

Parent process : 3
Parent process id :3256

Child process : 3
Child process id : 3257
Parent process id : 3256

Parent process : 4
Parent process id :3256

Child process : 4
Child process id : 3257
```

```
Parent process id : 3256

Parent process : 5
Parent process id :3256

Child process : 5
Child process id : 3257
Parent process id : 3256

Parent process : 6
Parent process id :3256

Child process : 6
Child process id : 3257
Parent process id : 3256

Parent process : 7
Parent process id :3256

Child process : 7
Child process id : 3257
Parent process id : 3256

Parent process : 8
Parent process id :3256

Child process : 8
Child process id : 3257
Parent process id : 3256

Parent process : 9
Parent process id :3256

Child process : 9
Child process id : 3257
```

```
Parent process id : 3256


Parent process : 10
Parent process id :3256

Parent process : 11
Parent process id :3256

Parent process : 12
Parent process id :3256

Parent process : 13
Parent process id :3256

Parent process : 14
Parent process id :3256

Parent process : 15
Parent process id :3256

Parent process : 16
Parent process id :3256

Parent process : 17
Parent process id :3256

Parent process : 18
Parent process id :3256

Parent process : 19
Parent process id :3256
```

Explore wait and waitpid before termination of process:

Code:

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

void main() {
     printf("PID: %4d,PPID: %4d\n",getpid(),getppid());
     printf("UID: %4d,GID: %4d\n",getuid(),getgid());
     printf("EUID: %4d,EGID: %4d\n",geteuid(),getegid());
     exit(0);
}
```

Output:

```
dan@home:~/Desktop$ chmod +x p4_3
dan@home:~/Desktop$ ./p4_3
PID: 3865, PPID: 3227
UID: 1000, GID: 1000
EUID: 1000, EGID: 1000
```

Code:

```
#include<fcntl.h>
#include<sys/stat.h>
#include<stdlib.h>
#define BUFSIZE 1024

int main(void) {
     int fd1,fd2;
     int n;
     char buf[BUFSIZE];
     fd1=open("old_file",O_RDONLY);
     fd2=open("new_file",
           O_WRONLY
           |O_CREAT
           |O_TRUNC,S_IRUSR
           |S_IWUSR
           |S_IRGRP
           |S_IWGRP
           |S_IROTH
     );

     while((n=read(fd1,buf,BUFSIZE))>0)
           write(fd2,buf,n);
     close(fd1);
     close(fd2);
     printf("File copied Successfully\n");
     exit (0);
}
```

Output:

```
dan@home:~/Desktop$ cat>old_file OLD FILE
dan@home:~/Desktop$ cat new_file
dan@home:~/Desktop$ gcc -o p4_2 p4_2.c
dan@home:~/Desktop$ ./p4_2 File copied Successfully
dan@home:~/Desktop$ cat new_file OLD FILE
```