Program using fork, getpid(), getppid(), close(), exit(), wait()

System calls

Practical: 3

Programs using the following system calls of Linux operating system: fork, getpid, getppid, exit, wait, close.

Getppid()

- 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. Otherwise, this function returns a value of 1 which is the process id for init process.
- Syntax:
- pid_t getppid(void);
- Return type: getppid() returns the process ID of the parent of the current process. It never throws any error therefore is always successful.

Getppid()

```
// C++ Code to demonstrate getppid()
#include <iostream>
#include <unistd.h>
using namespace std;

// Driver Code
int main()
{
    int pid;
    pid = fork();
    if (pid == 0)  {\\\ child process

            cout << "\nParent Process id : "<< getpid() << endl;
            cout << "\nChild Process with parent id : "<< getppid() << endl;
    }
    return 0;
}
```

Getpid()

- getpid() : returns the process ID of the calling process. This is often used by routines that generate unique temporary filenames.
- Syntax:
- pid_t getpid(void);
- Return type: getpid() returns the process ID of the current process. It never throws any error therefore is always successful.

Getpid()

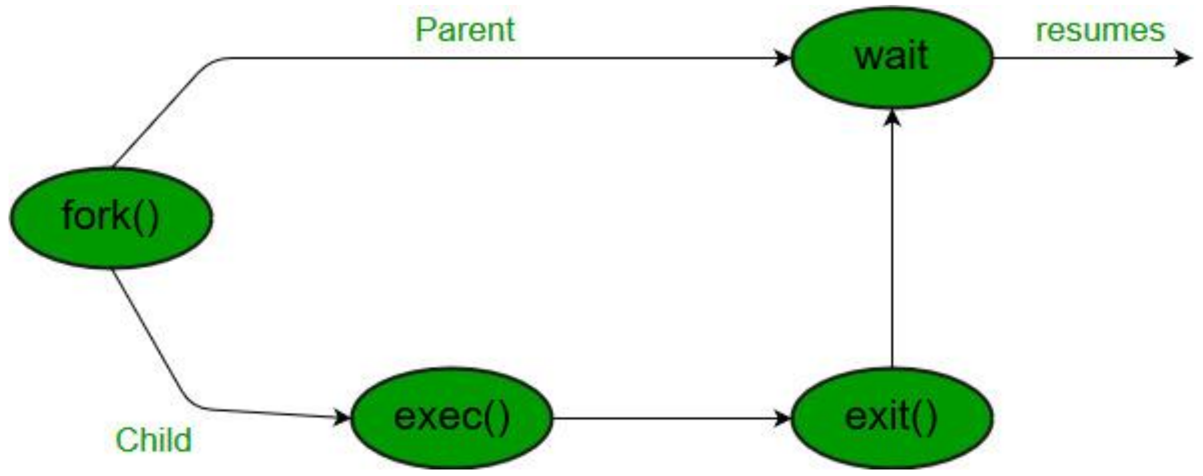
```
• // C++ Code to demonstrate getpid()
• #include <iostream>
• #include <unistd.h>
• using namespace std;

• // Driver Code
• int main()
• {
•     int pid = fork();
•     if (pid == 0)
•             cout << "\nCurrent process id of Process : "
•                     << getpid() << endl;
•     return 0;
• }
```

Wait System Call

- A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.
- Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.

Continued…



If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.

If only one child process is terminated, then return a wait() returns process ID of the terminated child process.

If more than one child processes are terminated than wait() reap any ***arbitrarily child*** and return a process ID of that child process.

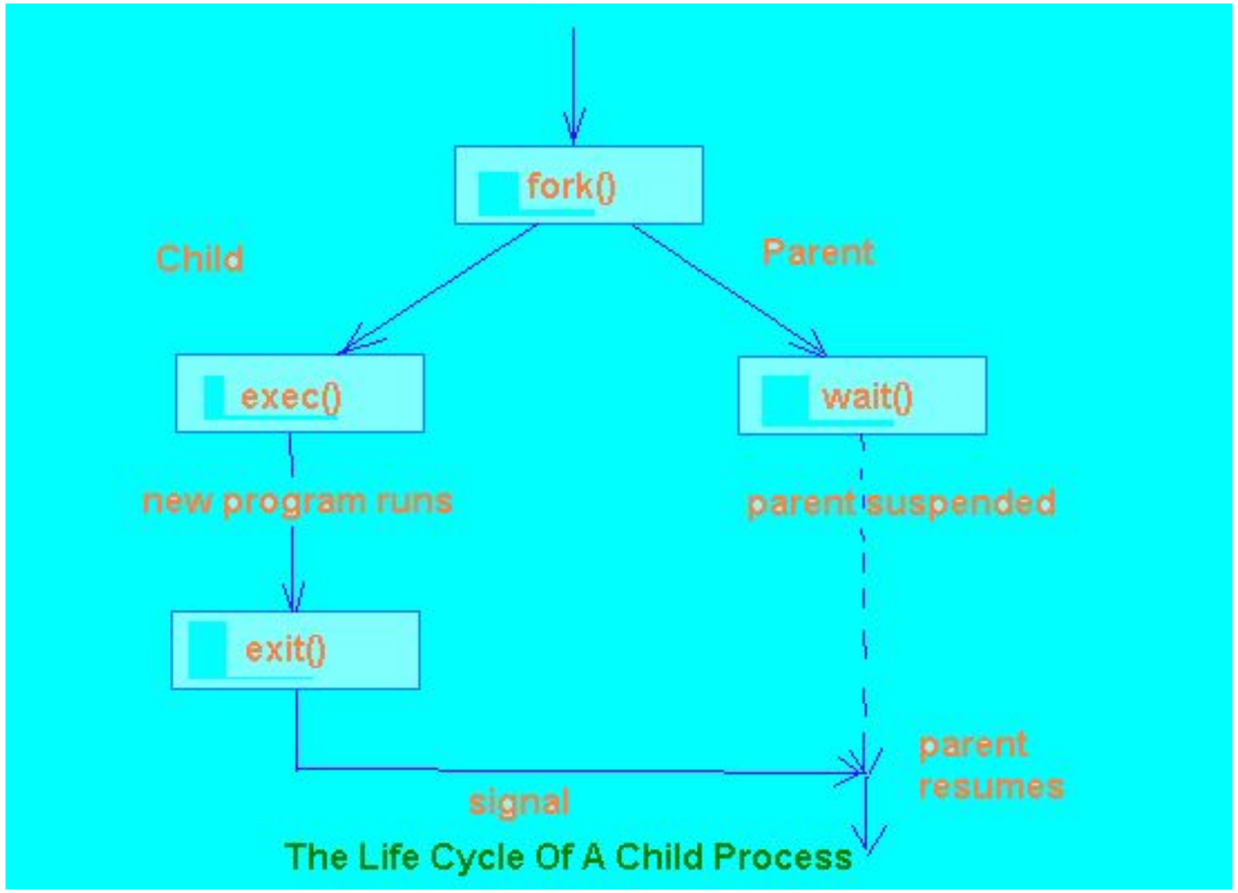
When wait() returns they also define **exit status** (which tells our, a process why terminated) via pointer, If status are not **NULL**.

If any process has no child process then wait() returns immediately "-1".

Program

```
// C program to demonstrate working of wait()
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    pid_t cpid;
    if (fork()== 0)
            exit(0);            /* terminate child */
    else
            cpid = wait(NULL); /* reaping parent */
    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);

    return 0;
}
```

Exit system call


fork()
Child
Parent
exec()
wait()
new program runs
parent suspended
exit()
parent
resumes
signal
The Life Cycle Of A Child Process