

EX. NO:5

DATE: 12.02.2025

### System Calls Programming

Aim:

To experiment system calls using fork(), execlp() and pid() functions.

Algorithm:

1. Start

o Include the required header files (stdio.h and stdlib.h).

2. Variable Declaration

o Declare an integer variable pid to hold the process ID.

3. Create a Process

o Call the fork() function to create a new process. Store the return value in the pid variable:

If fork() returns:

-1: Forking failed (child process not created).

0: Process is the child process.

Positive integer: Process is the parent process.

4. Print Statement Executed Twice

o Print the statement:

scss

Copy code

THIS LINE EXECUTED TWICE

(This line is executed by both parent and child processes after fork()).

5. Check for Process Creation Failure

o If pid == -1:

Print:

Copy code

CHILD PROCESS NOT CREATED

Exit the program using exit(0).

6. Child Process Execution

o If pid == 0 (child process):

Print:

Process ID of the child process using getpid().

Parent process ID of the child process using getppid().

7. Parent Process Execution

o If pid > 0 (parent process):

Print:

Process ID of the parent process using getpid().

Parent's parent process ID using getppid().

## 8. Final Print Statement

o Print the statement:

objectivec

33

Copy code

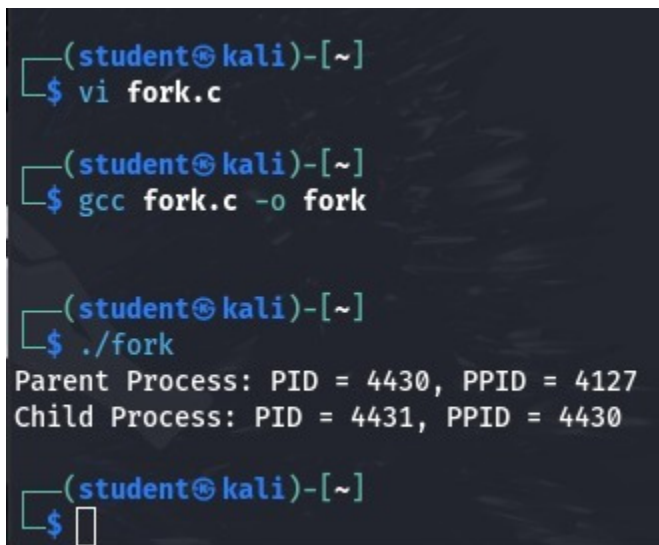
IT CAN BE EXECUTED TWICE

(This line is executed by both parent and child processes).

## 9. End

Program:

fork()



```
(student@kali)-[~]  
$ vi fork.c  
  
(student@kali)-[~]  
$ gcc fork.c -o fork  
  
(student@kali)-[~]  
$ ./fork  
Parent Process: PID = 4430, PPID = 4127  
Child Process: PID = 4431, PPID = 4430  
  
(student@kali)-[~]  
$
```

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

```
#include <stdlib.h>
```

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

```
int main() {
```

```
    int pid = fork();
```

```
    if (pid == -1) {
```

```
        printf("CHILD PROCESS NOT CREATED\n");
```

```
        exit(0);
```

```

    }
    if (pid == 0) {
        printf("Child Process: PID = %d, PPID = %d\n", getpid(), getppid());
    } else {
        printf("Parent Process: PID = %d, PPID = %d\n", getpid(), getppid());
    }
    return 0;
}

```

execlp()

```

(student@kali)-[~]
└─$ vi execlp.c

(student@kali)-[~]
└─$ gcc execlp.c -o execlp

(student@kali)-[~]
└─$ ./execlp
Before execlp()
total 184
drwxr-xr-x 2 student student 4096 Aug  6 2024 Desktop
drwxr-xr-x 2 student student 4096 Oct  1 09:18 Documents
drwxr-xr-x 3 student student 4096 Nov 12 08:52 Downloads
drwxr-xr-x 2 student student 4096 Jul 30 2024 Music
drwxr-xr-x 3 student student 4096 Oct 29 13:37 Pictures
drwxr-xr-x 2 student student 4096 Jul 30 2024 Public
drwxr-xr-x 2 student student 4096 Jul 30 2024 Templates
drwxr-xr-x 2 student student 4096 Jul 30 2024 Videos
-rw-rw-r-- 1 student student 348 Oct 29 13:28 WebScarab.properties
-rw-r--r-- 1 student student 2655 Aug  2 2024 arsath
-rw-r--r-- 1 student student 566 Aug  2 2024 arsath.pub
-rw-rw-r-- 1 student student 140 Feb  1 18:32 calc
-rw-rw-r-- 1 student student 131 Feb  1 18:37 calc.sh
-rw-r--r-- 1 student student  56 Feb  5 13:24 emp.dat.save
-rwxrwxr-x 1 student student 16008 Feb 12 08:55 execlp
-rw-rw-r-- 1 student student  354 Feb 12 08:55 execlp.c
-rwxrwxr-x 1 student student 16200 Feb 12 08:45 fork
-rw-rw-r-- 1 student student  490 Feb 12 08:45 fork.c
-rw-rw-r-- 1 student student  33 Aug 23 10:58 hashes.txt
-rw-rw-r-- 1 student student  0 Oct  1 09:03 hello.c
-rwxrwxr-x 1 student student 15960 Oct  1 09:09 helloworld
-rw-rw-r-- 1 student student  81 Oct  1 09:06 helloworld.c
-rw-r--r-- 1 student student 566 Aug  6 2024 helloworld.pub
-rw-rw-r-- 1 student student  12 Aug 13 2024 hi
-rw-rw-r-- 1 student student  0 Aug 13 2024 idrsa.hash
-rw-rw-r-- 1 student student 22727 Aug  6 2024 index.html
-rwxrwxr-x 1 student student 16400 Oct  1 09:24 injector
-rw-rw-r-- 1 student student 2063 Oct  1 09:24 injector.c
-rw-rw-r-- 1 student student  3 Aug 13 2024 rockyou.txt
-rw-rw-r-- 1 student student  7 Oct  1 09:50 shellcode.bin

```

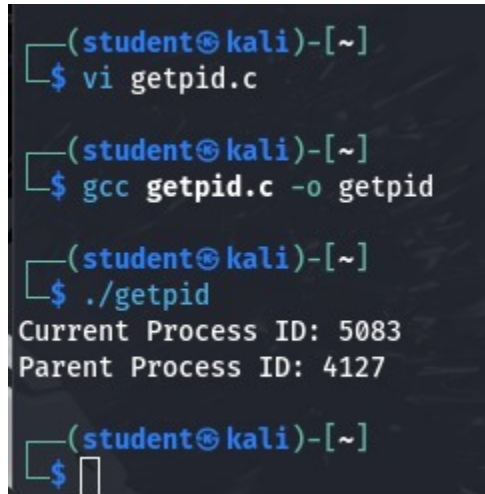
```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    printf("Before execlp()\n"); // Step 1: Print initial message
    execlp("ls", "ls", "-l", NULL); // Step 2: Execute "ls -l" command
    printf("This will not be printed if execlp() succeeds.\n"); // Step 3: This line is never
    // executed if execlp() works
}

```

```
    return 0;  
}
```

getpid()

A terminal window with a dark background and light blue text. The prompt is '(student@kali)-[~]'. The user enters '\$ vi getpid.c'. The prompt changes to '\$ gcc getpid.c -o getpid'. The user enters '\$ ./getpid'. The output is 'Current Process ID: 5083' followed by 'Parent Process ID: 4127'. The prompt returns to '\$' with a cursor.

```
(student@kali)-[~]  
$ vi getpid.c  
  
(student@kali)-[~]  
$ gcc getpid.c -o getpid  
  
(student@kali)-[~]  
$ ./getpid  
Current Process ID: 5083  
Parent Process ID: 4127  
  
(student@kali)-[~]  
$
```

```
#include <stdio.h>  
#include <unistd.h>  
int main() {  
    printf("Current Process ID: %d\n", getpid()); // Step 1: Get current PID  
    printf("Parent Process ID: %d\n", getppid()); // Step 2: Get parent process ID  
    return 0;  
}
```

opendir() and readdir()

```
(student@kali)-[~]
$ vi dir.c

(student@kali)-[~]
$ gcc dir.c -o dir

(student@kali)-[~]NT 5
$ ./dir
.msfx4
shellcode.bin
index.html
Videos
Downloads
.java
.zsh_history
dir.c
.bash_logout
.viminfo
hashes.txt
injector.c
Desktop
.ssh
WebScarab.properties
Templates
Documents
Music
getpid
.bashrc.original
calc.sh
.pki
emp.dat.save
execlp
fork
Public
helloworld
calc
.mozilla
getpid.c
.face.icon
.profile
arsath
helloworld.pub
dir
arsath.pub
.keystore
.face
.
injector
.john
..
.BurpSuite
.config
helloworld.c
.zshrc
.sudo_as_admin_successful
hello.c
execlp.c
hi
rockyou.txt
.cache
idrsa.hash
fork.c
.local
Pictures
.bashrc
```

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

```
#include <dirent.h>
```

```
int main() {
```

```
    struct dirent *de;
```

```
    DIR *dr = opendir("."); // Step 1: Open current directory
```

```
    if (dr == NULL) { // Step 2: Check for failure
```

```
        printf("Could not open current directory\n");
```

```
        return 0;
```

```
    }
```

```
    while ((de = readdir(dr)) != NULL) // Step 3: Read directory entries
```

```
        printf("%s\n", de->d_name);
```

```
    closedir(dr); // Step 4: Close directory
```

```
    return 0;
```

```
}
```

Result:

Hence, system calls are executed successfully.