

**VIT[®]****Vellore Institute of Technology**
(Deemed to be University under section 3 of UGC Act, 1956)

Lab 3:

Fork, Process, Signal and System Call


Programme	:	BTech. CSE Core	Semester	:	Win 2021-22
Course	:	Operating Systems	Code	:	CSE2005
Faculty	:	Dr. Shyamala L	Slot	:	L25+L26
Name	:	Hariket Sukesh Kumar Sheth	Register No.	:	20BCE1975

LAB 3

Aim: Create process and print parent ID and Child ID1

Steps:

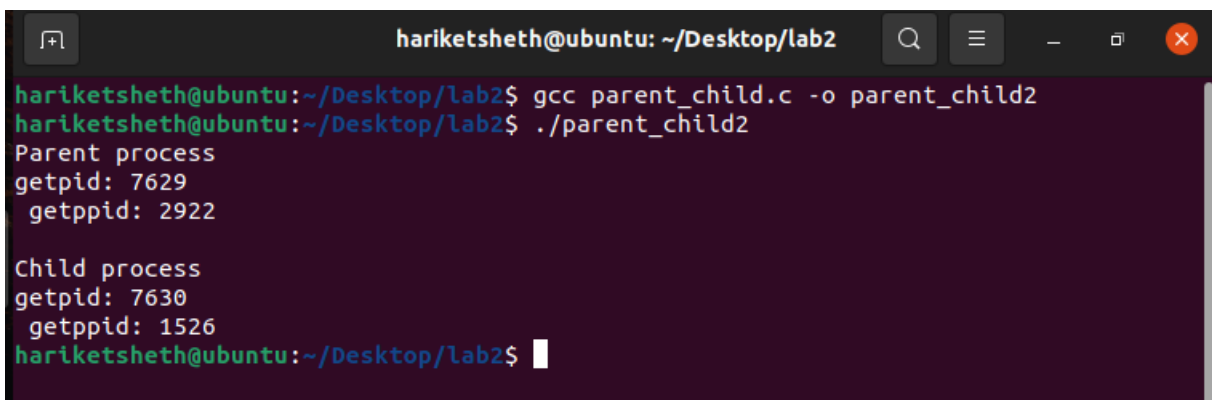
Step 1: Write the code for creating the process and printing the parent ID and child ID1



```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ cat parent_child.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(){
    int i;
    if(fork()==0){
        printf("Child process\n");
        printf("getpid: %d\n getppid: %d\n",getpid(),getppid());
        sleep(5);
    }
    else{
        printf("Parent process\n");
        printf("getpid: %d\n getppid: %d\n",getpid(),getppid());
        printf("\n");
    }
    return 0;
}
```

Step 2: Compile the code using gcc compiler and print the output in the terminal



```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ gcc parent_child.c -o parent_child2
hariketsheth@ubuntu:~/Desktop/lab2$ ./parent_child2
Parent process
getpid: 7629
getppid: 2922

Child process
getpid: 7630
getppid: 1526
hariketsheth@ubuntu:~/Desktop/lab2$
```

Aim: Create a process and compute factorial in child and Fibonacci in parent as executable

Steps:

Step 1: Write the code for Factorial printing in the child and Fibonacci number printing in the parent

```
fibonacci.c
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<stdlib.h>
4 #include<sys/types.h>
5 #include<sys/wait.h>
6
7 int main(){
8     int i,num;
9     if(fork()==0){
10         int i,fact=1,num;
11         printf("Enter number: ");
12         scanf("%d",&num);
13         for(i=1;i<=num;i++){
14             fact*=i;
15         }
16         printf("Factorial of %d is %d\n",num,fact);
17         exit(EXIT_SUCCESS);
18     }
19     else{
20         wait(NULL);
21         int n1=0,n2=1,n3,i,num;
22         printf("Enter number: ");
23         scanf("%d",&num);
24         printf("\n%d %d ",n1,n2);
25         for(i=2;i<=num;i++){
26             n3=n1+n2;
27             printf("%d ",n3);
28             n1=n2;
29             n2=n3;
30         }
31         printf("\n");
32     }
33     return 0;
34 }
35
```

Step 2: Compile the code using gcc compiler and print the output in the terminal

```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ ./fibonacci
Enter number: 5
Factorial of 5 is 120
Enter number: 9
0 1 1 2 3 5 8 13 21
```

Aim: Program to create four processes (1 parent and 3 children) where they terminates in a sequence as follows :

- (a) Parent process terminates at last
- (b) First child terminates before parent and after second child.
- (c) Second child terminates after last and before first child.
- (d) Third child terminates first.

```
even_odd.c      x      fibo_fact.c      x      *fork_process.c
1#include <stdio.h>
2#include <stdlib.h>
3#include <unistd.h>
4
5int main(){
6    int pid, pid_1, pid_2;
7    pid = fork();
8    if (pid == 0) {
9        sleep(1);
10       printf("Child Process 1 \npID: %d \nppID: %d\n", getpid(), getppid());
11    }
12
13    else {
14        pid_1 = fork();
15        if (pid_1 == 0) {
16            sleep(2);
17            printf("Child Process 2 \npID: %d \nppID: %d\n", getpid(), getppid());
18        }
19        else {
20            pid_2 = fork();
21            if (pid_2 == 0)
22                printf("Child Process 3 \npID: %d \nppID: %d\n", getpid(), getppid());
23
24            else {
25                sleep(4);
26                printf("Parent Process \npID: %d\n", getpid());
27            }
28        }
29    }
30
31    return 0;
32 }
33
```

```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ gcc fork_process.c -o fork2
hariketsheth@ubuntu:~/Desktop/lab2$ ./fork2
Child Process 3
pID: 18259
ppID: 18256
Child Process 2
pID: 18258
ppID: 18256
Parent Process
pID: 18256
Child Process 1
pID: 18257
ppID: 18256
hariketsheth@ubuntu:~/Desktop/lab2$
```

Date: 04-02-2022

LAB 03

Fork and System Call



LAB 3

Aim: Write a C program to create a Child process using **fork** system call to print even numbers and parent prints odd number till 50.

Steps:

Step 1: Write the code for printing the odd numbers (Parent process) and even numbers (child Process) till 50 using fork

Step 2: Compile the code and print the output (without modification)

```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ gcc even_odd.c -o even_odd2
hariketsheth@ubuntu:~/Desktop/lab2$ ./even_odd2
odd 1
odd 3
odd 5
odd 7
odd 9
odd 11
odd 13
odd 15
odd 17
odd 19
odd 21
odd 23
odd 25
odd 27
odd 29
odd 31
odd 33
odd 35
odd 37
odd 39
odd 41
odd 43
odd 45
odd 47
odd 49
even 2
even 4
even 6
even 8
even 10
even 12
even 14
even 16
even 18
even 20
even 22
even 24
even 26
even 28
even 30
even 32
even 34
even 36
even 38
even 40
even 42
even 44
even 46
even 48
```

1. **The parent and child are two processes; can we say parent runs first always? If so, do scheduling and context switching have any impact on them?**

It's difficult to tell which process runs first: the parent or the child. They can operate on multiple CPU cores, allowing them to execute in parallel, or they can run on the same core. The parent and child processes will effectively operate at the same time after a successful `fork()`. These processes' output can come in the output stream at any moment.

2. **In multi-processor environment, how will you ensure the parent controls child process,**

A child process will be established after `fork()`.

The parent received the process id of the child as the return value of `fork()` and will output odd numbers, whereas the child received the return value of `fork()` and will print even numbers.

CPU Scheduling occurs when numerous processes execute on the same CPU core. It's impossible for a CPU core to accomplish two things at once.

3. **How will you ensure child finishes first and then parent?**

Although the parent and child processes will finish at roughly the same time in this example, it is generally a good practise for a parent process that forks a child process to execute a subsequent wait call (or `signal(SIGCHLD, SIG_IGN)`; to indicate to the kernel that the child's exit status is not required).

When using `vfork`, virtually all implementations specify that the child shall run first, followed by the parent. (Until the child calls `exec`) As a result, regardless of scheduler, serial execution will be seen in the case of `vfork`.

Step 3: Modify the code to execute the child process first and then the parent process

```
*even_odd.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7
8 int main()
9 {
10     int i, status;
11     pid_t pid;
12
13
14     pid = fork();
15     if (pid == 0)
16     {
17         for(i=1; i<50; i++)
18             if(i%2 == 0)
19                 printf("even %d\n", i);
20     }
21     else
22     {
23         if (pid > 0)
24             pid = waitpid(pid, &status, 0);
25         for(i=1; i<50; i++)
26             if(i%2 != 0)
27                 printf("odd %d\n", i);
28     }
29     return 0;
30 }
31
```

Step 4: Compile the code and print the output (with modification)

```
hariketsheth@ubuntu: ~/Desktop/lab2
hariketsheth@ubuntu:~/Desktop/lab2$ gcc even_odd.c -o even_odd2
hariketsheth@ubuntu:~/Desktop/lab2$ ./even_odd2
even 2
even 4
even 6
even 8
even 10
even 12
even 14
even 16
even 18
even 20
even 22
even 24
even 26
even 28
even 30
even 32
even 34
even 36
even 38
even 40
even 42
even 44
even 46
even 48
odd 1
odd 3
odd 5
odd 7
odd 9
odd 11
odd 13
odd 15
odd 17
odd 19
odd 21
odd 23
odd 25
odd 27
odd 29
odd 31
odd 33
odd 35
odd 37
odd 39
odd 41
odd 43
odd 45
odd 47
odd 49
```

Codes:**- parent_child.c**

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

int main(){
    int i;
    if(fork()==0){
        printf("Child process\n");
        printf("getpid: %d\n getppid: %d\n",getpid(),getppid());
        sleep(5);
    }
    else{
        printf("Parent process\n");
        printf("getpid: %d\n getppid: %d\n",getpid(),getppid());
        printf("\n");
    }
    return 0;
}
```

- fibo_fact.c

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

int main(){
    int i,num;
    if(fork()==0){
        int i,fact=1,num;
        printf("Enter number: ");
        scanf("%d",&num);
        for(i=1;i<=num;i++)
            fact*=i;
        printf("Factorial of %d is %d\n",num,fact);
        exit(EXIT_SUCCESS);
    }

    else{
        wait(NULL);
        int n1=0,n2=1,n3,i,num;
    }
}
```



```
printf("Enter number: ");
scanf("%d",&num);
printf("\n%d %d ",n1,n2);
for(i=2;i<num;++i){
    n3=n1+n2;
    printf("%d ",n3);
    n1=n2;
    n2=n3;
}
printf("\n");
}
return 0;
}
```

- fork_process.c

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

int main(){
    int pid, pid_1, pid_2;
    pid = fork();
    if (pid == 0) {
        sleep(3);
        printf("Child Process 1 \npID: %d \nppID: %d\n",getpid(), getppid());
    }

    else {
        pid_1 = fork();
        if (pid_1 == 0) {
            sleep(2);
            printf("Child Process 2 \npID: %d \nppID: %d\n", getpid(), getppid());
        }
        else {
            pid_2 = fork();
            if (pid_2 == 0)
                printf("Child Process 3 \npID: %d \nppID: %d\n", getpid(), getppid());

            else {
                sleep(3);
                printf("Parent Process \npID: %d\n", getpid());
            }
        }
    }
}
```

```
    return 0;
}
```

- even_odd.c

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

int main()
{
    int i, status;
    pid_t pid;

    pid = fork();
    if (pid == 0)
    {
        for(i=1;i<50;i++)
            if(i%2 == 0)
                printf("even %d\n",i);
    }
    else
    {
        if (pid > 0)
            pid = waitpid(pid, &status, 0);
        for(i=1;i<50;i++)
            if(i%2 != 0)
                printf("odd %d\n",i);
    }
    return 0;
}
```