

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

LAB EXPERIMENT 2

Process Creation

1. Create process and print parent ID and Child ID

Program:

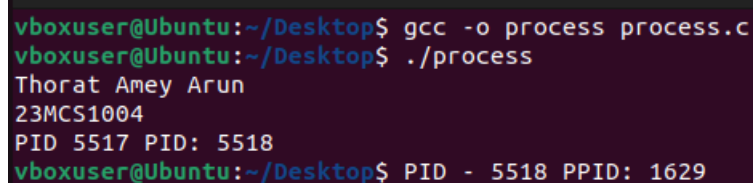
```
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("Thorat Amey Arun\n23MCS1004");
    pid_t pid = fork(); // create a new process

    if (pid == -1)
        printf("Failed to create a new process\n");
    else if (pid == 0)
        printf("PID - %d PPID: %d\n", getpid(), getppid());
    else
        printf("PID %d PID: %d\n", getpid(), pid);

    return 0;
}
```

Output:



```
vboxuser@Ubuntu:~/Desktop$ gcc -o process process.c
vboxuser@Ubuntu:~/Desktop$ ./process
Thorat Amey Arun
23MCS1004
PID 5517 PID: 5518
vboxuser@Ubuntu:~/Desktop$ PID - 5518 PPID: 1629
```

2. Create a process and compute factorial in child and Fibonacci in parent
 - a. As separate code in child

Program:

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

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
#include <unistd.h>
#include <sys/wait.h>
// Function to compute factorial
unsigned long long factorial(int n) {
    if (n <= 1)
        return 1;
    return n * factorial(n - 1);
}
// Function to compute Fibonacci
unsigned long long fibonacci(int n) {
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n;
    printf("Thorat Amey Arun\n23MCS1004");
    printf("Enter a number: ");
    scanf("%d", &n);
    pid_t pid = fork();
    if (pid == -1) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) { // Child process
        // Compute factorial
        unsigned long long fact = factorial(n);
        printf("Child process: Factorial of %d is %llu\n", n, fact);
    } else { // Parent process
        wait(NULL); // Wait for the child process to complete
        // Compute Fibonacci
        unsigned long long fib = fibonacci(n);
        printf("Parent process: Fibonacci of %llu\n", fib);
        for (int i = 0; i < n; i++) {
            printf("%llu ", fibonacci(i));
        }
        printf("\n");
    }
}
```

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
}  
return 0;  
}
```

Output:

```
vboxuser@Ubuntu:~/Desktop$ gcc -o factc_fibop fibonacci.c  
vboxuser@Ubuntu:~/Desktop$ ./factc_fibop  
Thorat Amey Arun  
23MCS1004  
Enter a number: 23  
Child process: Factorial of 23 is 8128291617894825984  
Parent process: Fibonacci of 28657 :0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711  
vboxuser@Ubuntu:~/Desktop$
```

b. As executable file called in child.

Program:

fc.c:

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

```
unsigned long long factorial(int n) {  
    if (n == 0 || n == 1) {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}
```

```
int main() {  
    int n = 5; // Factorial of 5  
    pid_t pid = fork();  
  
    if (pid < 0) {  
        fprintf(stderr, "Fork failed\n");  
        return 1;  
    } else if (pid == 0) {  
        // Child process  
        unsigned long long fact = factorial(n);
```

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
        printf("Factorial of %d is %llu\n", n, fact);
    } else {
        // Parent process
        wait(NULL); // Wait for child process to complete
        printf("Child process completed\n");
    }

    return 0;
}
```

fp.c:

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

```
int fibonacci(int n) {
    if (n <= 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

```
int main() {
    int n = 10; // Calculate Fibonacci sequence up to the 10th element
    pid_t pid = fork();

    if (pid < 0) {
        fprintf(stderr, "Fork failed\n");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Fibonacci sequence: ");
        for (int i = 0; i < n; i++) {
            printf("%d ", fibonacci(i));
        }
        printf("\n");
    } else {

```

OPERATING SYSTEM LAB

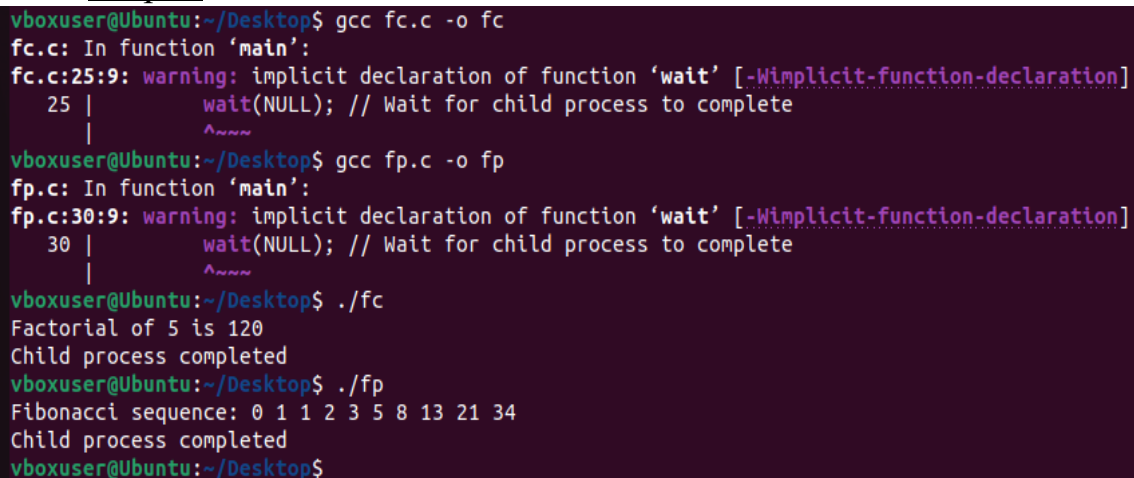
Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
// Parent process
wait(NULL); // Wait for child process to complete
printf("Child process completed\n");
}

return 0;
}
```

Output:



```
vboxuser@Ubuntu:~/Desktop$ gcc fc.c -o fc
fc.c: In function 'main':
fc.c:25:9: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
 25 |         wait(NULL); // Wait for child process to complete
    |         ^~~~~
vboxuser@Ubuntu:~/Desktop$ gcc fp.c -o fp
fp.c: In function 'main':
fp.c:30:9: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
 30 |         wait(NULL); // Wait for child process to complete
    |         ^~~~~
vboxuser@Ubuntu:~/Desktop$ ./fc
Factorial of 5 is 120
Child process completed
vboxuser@Ubuntu:~/Desktop$ ./fp
Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34
Child process completed
vboxuser@Ubuntu:~/Desktop$
```

3. 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

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
printf("Thorat Amey Arun\n23MCS1004");
pid_t pid[3];
```

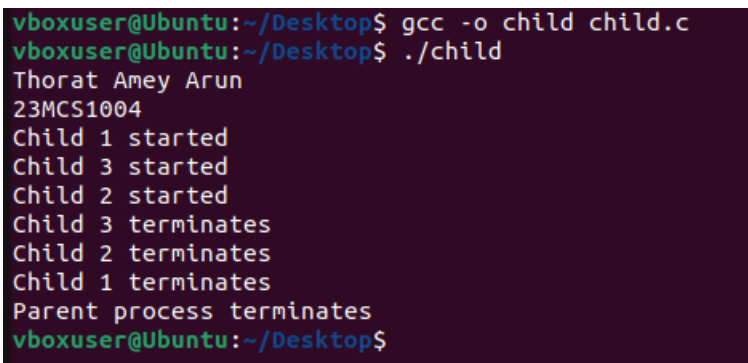
OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
// Create three child processes
for (int i = 0; i < 3; i++) {
    pid[i] = fork();
    if (pid[i] == -1) {
        perror("Fork failed");
        exit(1);
    } else if (pid[i] == 0) { // Child process
        printf("Child %d started\n", i + 1);
        if (i == 2) {
            printf("Child 3 terminates\n");
            exit(0);
        } else {
            sleep(3 - i);
            printf("Child %d terminates\n", i + 1);
            exit(0);
        }
    }
}
// Parent process
for (int i = 0; i < 3; i++) {
    waitpid(pid[i], NULL, 0);
}
printf("Parent process terminates\n");
return 0;
}
```

Output:



```
vboxuser@Ubuntu:~/Desktop$ gcc -o child child.c
vboxuser@Ubuntu:~/Desktop$ ./child
Thorat Amey Arun
23MCS1004
Child 1 started
Child 3 started
Child 2 started
Child 3 terminates
Child 2 terminates
Child 1 terminates
Parent process terminates
vboxuser@Ubuntu:~/Desktop$
```

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

4. Create Zombie and orphan process and analyse its behaviour and discuss how to avoid that.

➤ ZOMBIE PROCESS

A zombie process is a process that has completed its execution but still has an entry in the process table. This can happen when the parent process doesn't call the wait() or waitpid() function to collect the exit status of its child process.

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
//Note: Zombie process is a process which has terminated but it's entry is
there in the system.
//NOTE:To check the pid in linux cmd : "ps -e -o pid,stat"
int main() {
    printf("Thorat Amey Arun\n23MCS1004");
    int pid=fork();
    if(pid == 0)
    {
        printf("child process id : %d has parent id : %d \n", getpid(),
        getppid());
    }
    else if(pid > 0)
    {
        sleep(60); //Putting parent into sleep so that child get executed.
        printf("parent process id: %d has grand parent id : %d \n",
        getpid(), getppid());
    }
    else
    {
        printf("process not created");
    }
    return 0;
}
```

Output:

In the output we can observe that after the Parent Process execution has been completed, child process is still showing the process table.

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
vboxuser@Ubuntu:~/Desktop$ gcc zombie.c
vboxuser@Ubuntu:~/Desktop$ ./a.out &
[1] 6103
vboxuser@Ubuntu:~/Desktop$ Thorat Amey Arun
23MCS1004
child process id : 6104 has parent id : 6103
ps
  PID TTY          TIME CMD
  4777 pts/0        00:00:00 bash
   6103 pts/0        00:00:00 a.out
   6104 pts/0        00:00:00 a.out <defunct>
   6105 pts/0        00:00:00 ps
vboxuser@Ubuntu:~/Desktop$ ps
  PID TTY          TIME CMD
  4777 pts/0        00:00:00 bash
   6103 pts/0        00:00:00 a.out
   6104 pts/0        00:00:00 a.out <defunct>
   6109 pts/0        00:00:00 ps
vboxuser@Ubuntu:~/Desktop$ parent process id: 6103 has grand parent id : 4777
```

➤ ORPHAN Process

An orphan process is a process whose parent process terminates before it does. In this case, the orphan process gets adopted by the init process (with PID 1).

Program:

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

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

```
#include <sys/types.h>
```

```
int main()
```

```
{
```

//Note: to run the program use "./<filename>.out &" so that program will run in the background we can check the process running using ps

command.

```
printf("Thorat Amey Arun\n23MCS1004");
```

```
pid_t p;
```

```
p=fork();
```

```
if(p==0)
```

```
{
```

```
sleep(10); //putting child process to sleep so that parent will be executed
```

```
printf("I am child having PID %d\n",getpid());
```

```
printf("My parent PID is %d\n",getppid());
```

```
}
```

```
else
```


OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
{
sleep(2);
printf("I am parent having PID %d\n",getpid());
printf("My child PID is %d\n",p);
}
}
```

Output:

Here in the output we can observe that parent PID : 24942 and child PID : 24943 , child process is put to sleep and at that time parent process is executed, after resuming back child process has no parent since it is already executed. In the last line we can see PID of parent is 1714 which is different from original parent id 24942.

```
vboxuser@Ubuntu:~/Desktop$ gcc orphan.c
vboxuser@Ubuntu:~/Desktop$ ./a.out &
[1] 5960
vboxuser@Ubuntu:~/Desktop$ Thorat Amey Arun
23MCS1004
I am parent having PID 5960
My child PID is 5961
ps
  PID TTY          TIME CMD
  4777 pts/0        00:00:00 bash
  5961 pts/0        00:00:00 a.out
  5965 pts/0        00:00:00 ps
[1]+  Done                  ./a.out
vboxuser@Ubuntu:~/Desktop$ I am child having PID 5961
My parent PID is 1629
ps
  PID TTY          TIME CMD
  4777 pts/0        00:00:00 bash
  5966 pts/0        00:00:00 ps
vboxuser@Ubuntu:~/Desktop$
```

Analyzing the Behavior:

- In the first program (zombie process), the parent process doesn't wait for the child process to complete. The child process sleeps for 10 seconds and then exits. During this time, the child process becomes a zombie, as it has completed execution but its exit status is still retained by the process table. The zombie process is only cleared once the parent process exits.
- In the second program (orphan process), the parent process exits before the child process does. The child process continues execution, even after the parent exits. It gets adopted by the init process (with PID 1714).

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

Avoiding Zombie and Orphan Processes:

➤ Reaping Zombie Processes:

In the case of zombie processes, the parent process should call the wait() or waitpid() function to collect the exit status of its child processes. This way, the system cleans up the resources associated with the child process.

➤ Preventing Orphan Processes:

To avoid orphan processes, parents should ensure they don't exit before their child processes. Proper synchronization mechanisms like waiting for child processes to complete can help prevent orphans.

Here's how you could modify the first program to avoid creating a zombie process:

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
printf("Name: Vishal Papan\nRegno: 23MCS1011\n");
pid_t child_pid = fork();
if (child_pid == 0) { // Child process
printf("Child process is sleeping\n");
sleep(5);
printf("Child process completed\n");
} else if (child_pid > 0) { // Parent process
printf("Parent process created a child with PID: %d\n",
child_pid);
wait(NULL); // Wait for the child to complete
printf("Parent process completed\n");
} else {
perror("Fork failed");
exit(1);
}
return 0;
}
```

Output:

OPERATING SYSTEM LAB

Name: Thorat Amey Arun

Reg No.: 23MCS1004

```
vboxuser@Ubuntu:~/Desktop$ gcc zo.c
[1]+  Done                  ./a.out
vboxuser@Ubuntu:~/Desktop$ ./a.out &
[1] 6155
vboxuser@Ubuntu:~/Desktop$ Thorat Amey Arun
23MCS1004
Parent process created a child with PID: 6156
Child process is sleeping
Child process completed
Parent process completed
ps
  PID TTY          TIME CMD
  4777 pts/0        00:00:00 bash
  6159 pts/0        00:00:00 ps
[1]+  Done                  ./a.out
vboxuser@Ubuntu:~/Desktop$
```