

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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:**Fork, Process, Signal and System Call

**1**

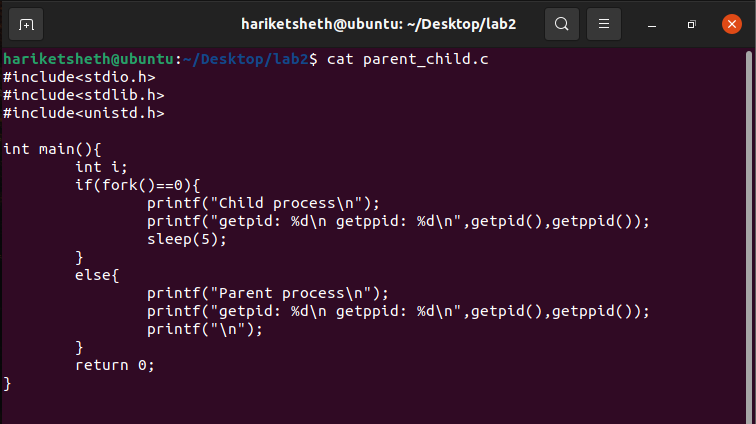
|  |  |  |  |
| --- | --- | --- | --- |
| Date: 04-02-2022 | LAB 03 | Fork and System Call |  |

**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



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

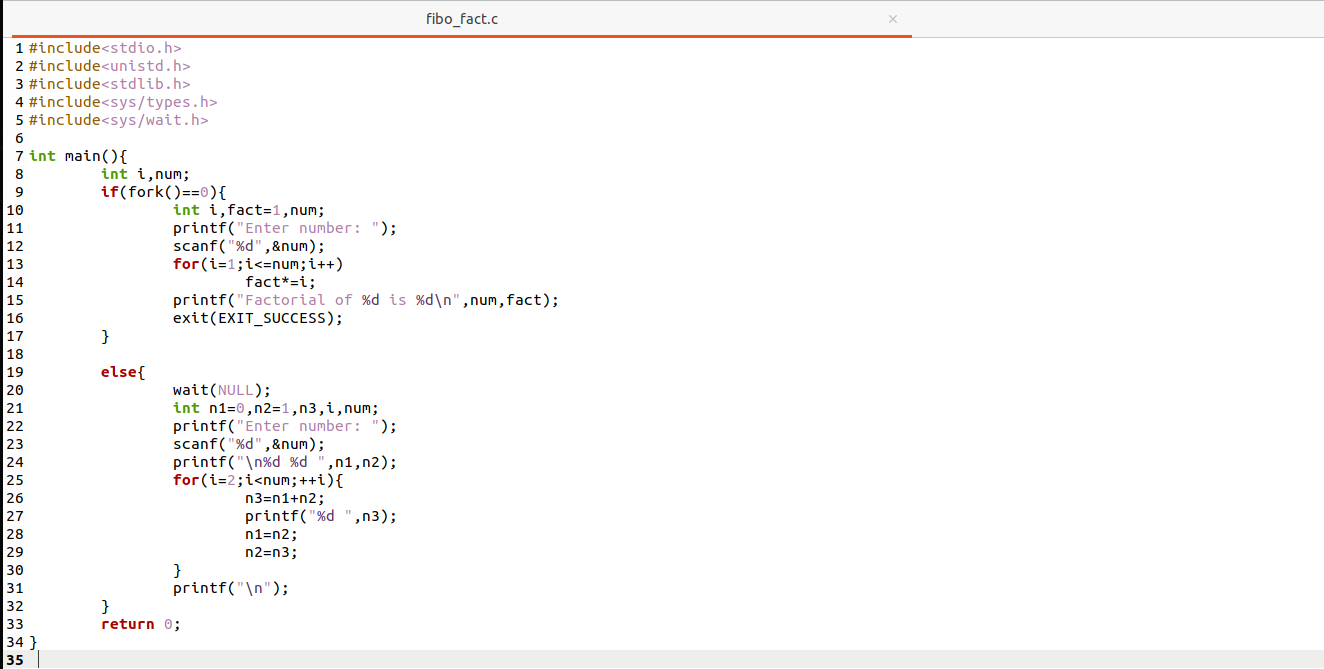


**2**

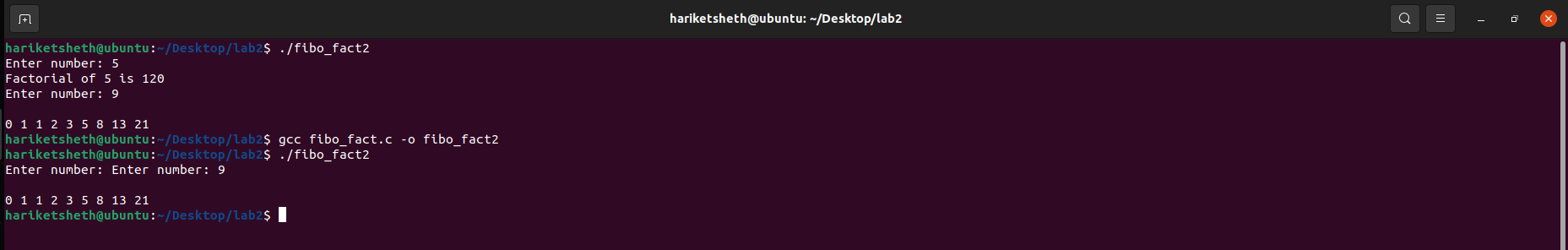
**Aim:** Create a process and compute factorial in child and Fibonacci in parent as executable

**Steps:**

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

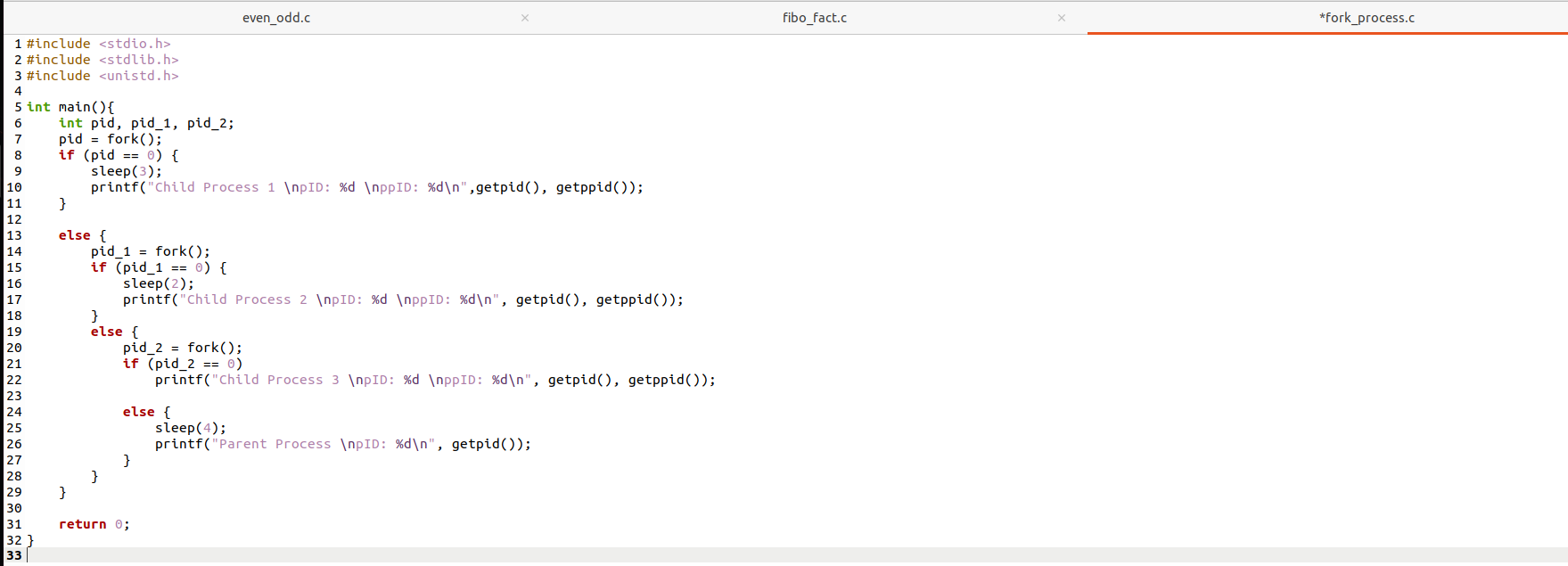


**Step 2:** Compile the code using gcc compiler and print the output in the terminal

****

**3**

**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.



**4**

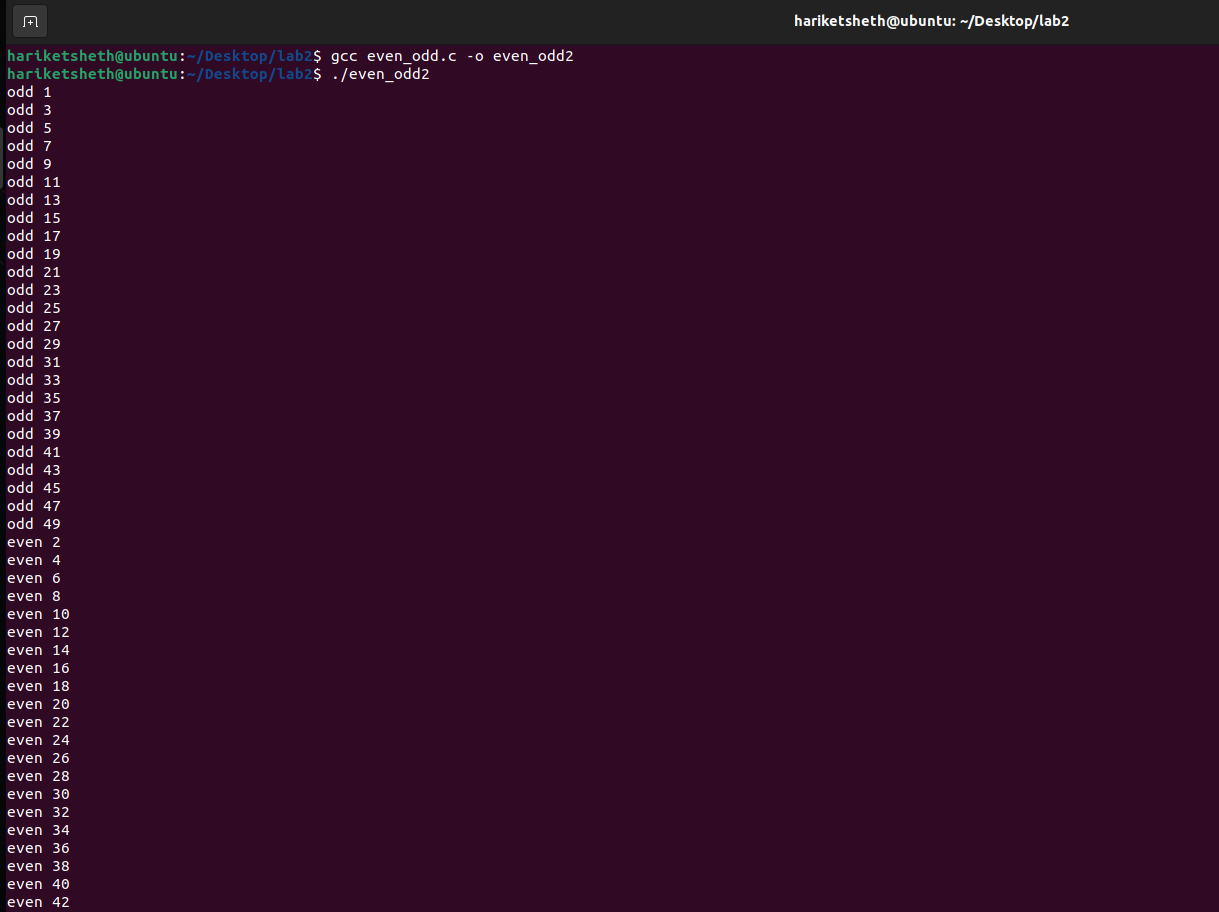
|  |  |  |  |
| --- | --- | --- | --- |
| Date: 04-02-2022 | LAB 03 | Fork and System Call |  |

**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)

**5**

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.

1. **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.

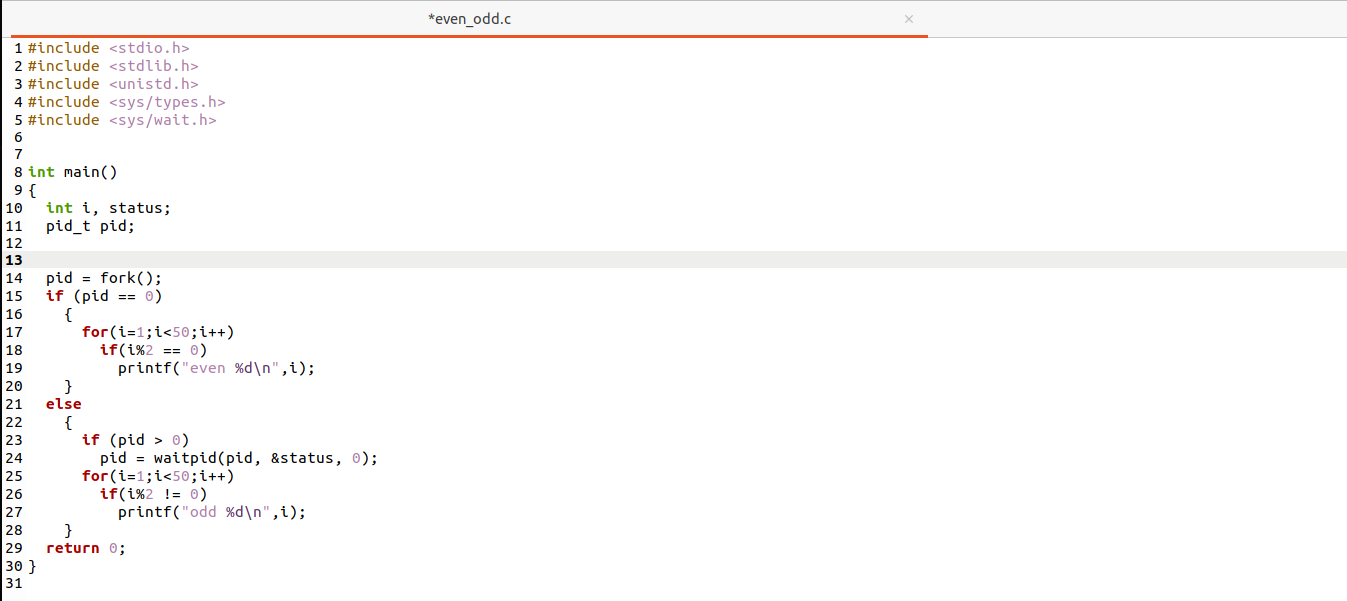
1. **How will you ensure child finishes first and then parent?**

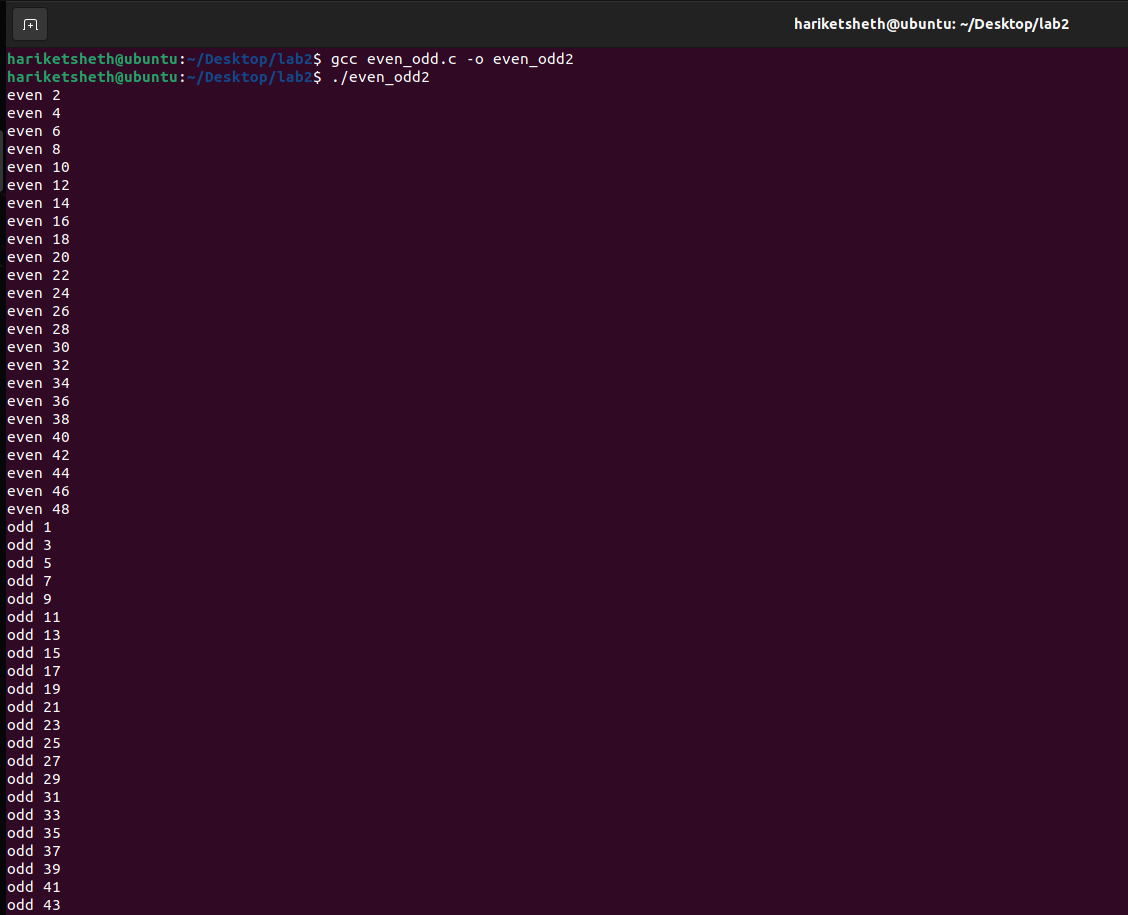
Although the parent and child processes will finish at roughly the same time in this example, it is generally a good practise for a parent process that forks a child process to execute a subsequent wait call (or **signal(SIGCHLD, SIG IGN);** to indicate to the kernel that the child's exit status is not 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 schedular, serial execution will be seen in the case of **vfork**.

**6**

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



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

**7**

**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);**

**8**

**}**

**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 {**

**9**

**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;**

**}**

**10**