**Processes in Linux**

**Lab no# 03**

****

**Fall 2021**

**CSE-302 System Programming Lab**

Submitted by: **Ashfaq Ahmad**

Registration No: **19PWCSE1795**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Engr: Ma’am Madeha sheer**

**November** 28, 2021

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Task 1:** Create process chain as shown in figure 3.1(b) and fill the figure 3.1 (b) with actual IDs. The program shall take a single command-line argument that specifies the number of processes to be created. Before exiting, each process shall outputs its i value (loop variable), its process ID (using getpid()), its parent process ID (getppid()) and the process ID of its child (return value of fork). The parent does not execute wait.

**Source code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int x;

for(int i=0; i<4; i++)

{

x=fork();

if (x>0) //condition for chain process.

{

printf("PID: %d PPID: %d Child PID: %d\n",getpid(),getppid(),x);

break;

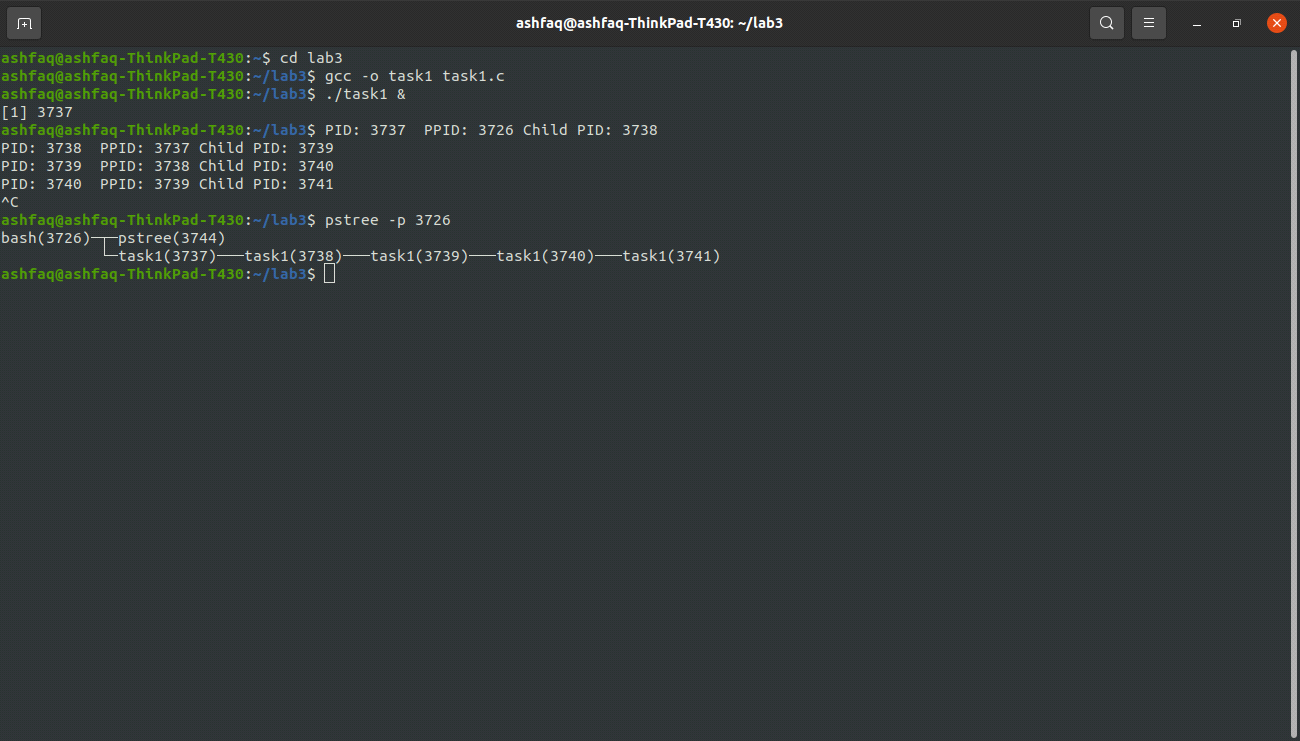
}

}

while(1);

}

**Output:**



**Task 2:** Create process fan as shown in figure 3.1 (a) and fill the figure 3.1 (a) with actual IDs.

**Source code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int x;

for(int i=0; i<7; i++)

{

x=fork();

if(x==0) //condition for fan process

{

printf("PID: %d PPID: %d\n",getpid(),getppid());

break;

}

}

while(1);

}

**Output:**



**Task 3:** Create process tree as shown in figure 3.2 and fill figure 3.2 with actual IDs.

**Source code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int x;

for(int i=0; i<4; i++)

{

x=fork(); //here is no condition for process tree.

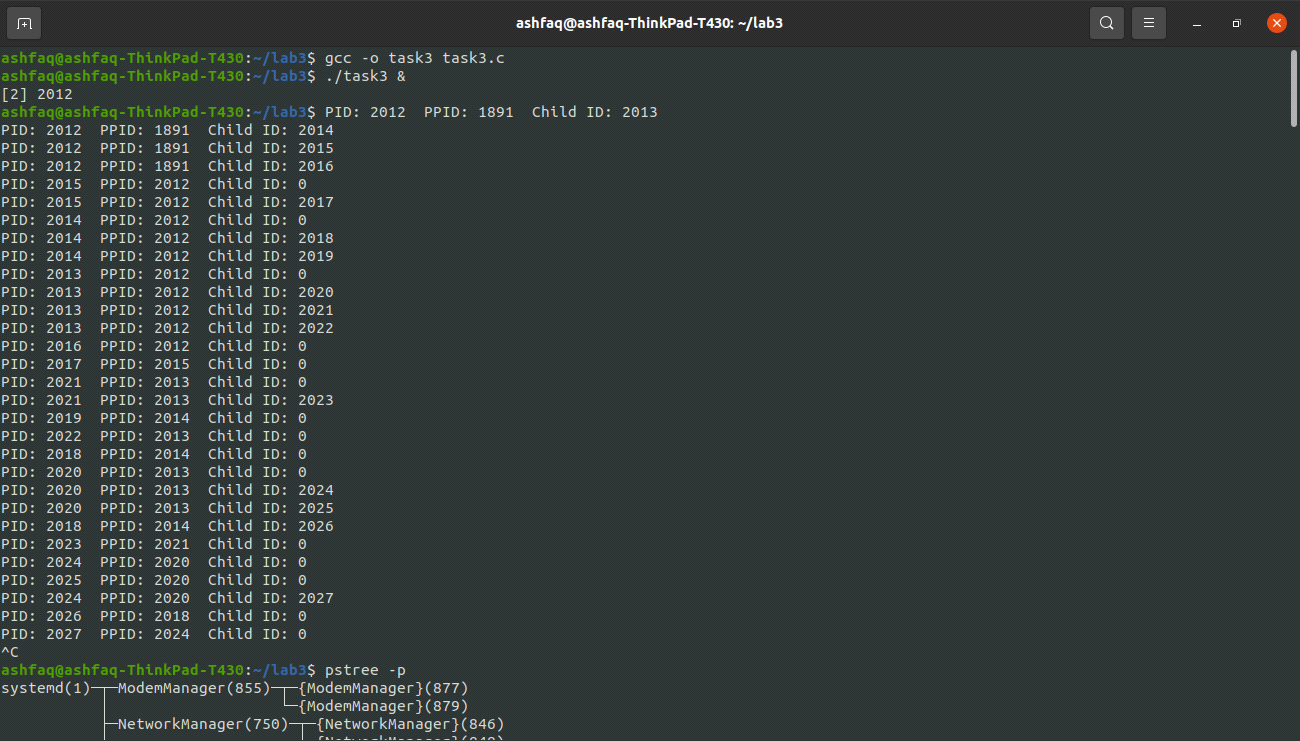
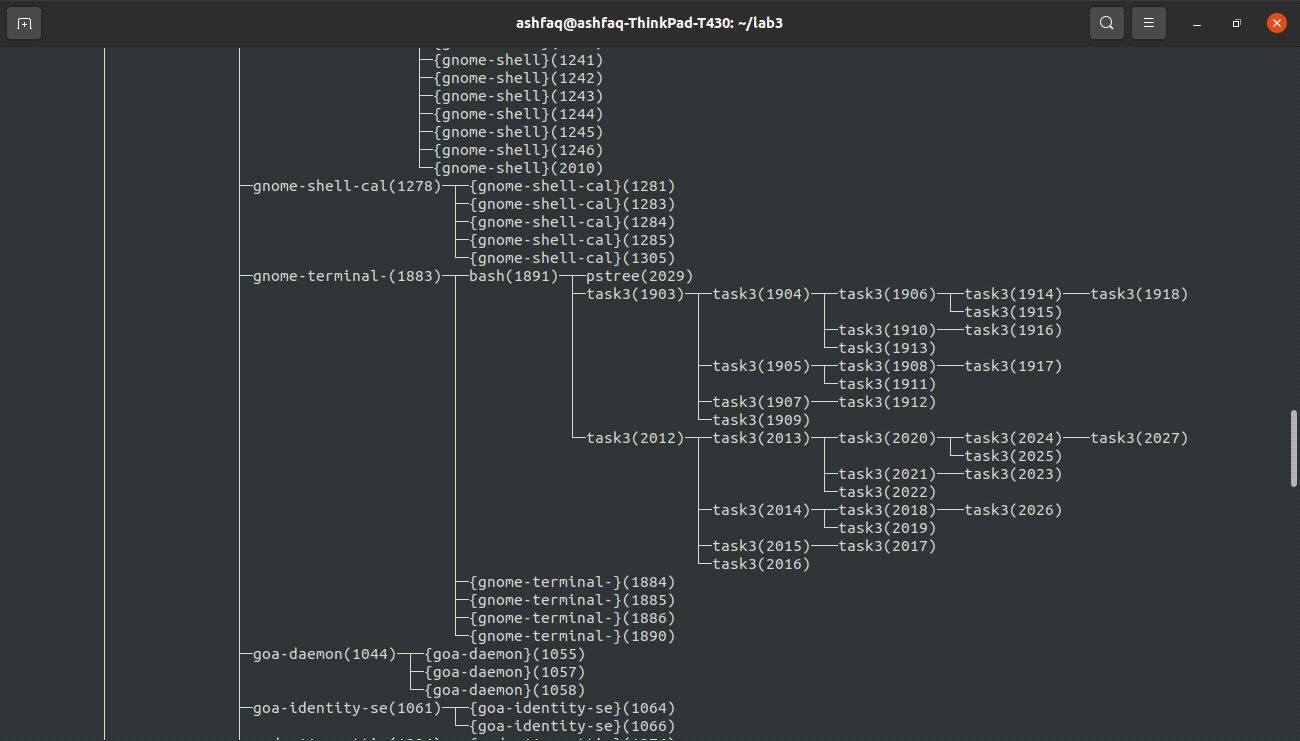
printf("PID: %d PPID: %d\n",getpid(),getppid());

}

while(1);

}

**Output:**



**Task 4:** This task expands on the process chain of Task 1. The chain is a vehicle for experimenting with wait and with sharing of devices. All of the processes in the chain created by Task 1 share standard input, standard output and standard error.

Task 3.1 creates a chain of processes. It takes a single command-line argument that specifies the number of processes to create. Before exiting, each process outputs its i value, its process ID, its parent process ID and the process ID of its child. The parent does not execute wait. If the parent exits before the child, the child becomes an orphan. In this case, the child process is adopted by a special system process (which traditionally is a process, init, with process ID of 1). As a result, some of the processes may indicate a parent process ID of 1.

1. Experiment with different values for the command-line argument. Observe the fractions that are adopted by init.
2. Place sleep(10); directly before the final printf statement. What is the maximum number of processes generated in this case?

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int x;

for (int i=0; i<4; i++)

{

x=fork();

if(x==-1)

{

perror("error occured");

return 0;

}

if (x>0)

{

printf("PID: %d PPID: %d Child ID: %d", getpid(),getppid(),x);

break;

}

}

}

**Discussion:**

1. Without using wait/sleep function almost all the process become orphan and we adopted by the init process. And we can make a chain of as many processes as we want.
2. After putting the sleep function in our code, the maximum no of child process generated reduces to 3318 on my system, after 3318 child processes are created the OS crashes if we try to fork another child.

**Task 5:** Write a program that takes N number of integers as argument and displays the factors of N integers. Create separate child process for each integer. Make sure no child is orphan/zombie.

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

void factor(int x)

{

for(int i=1; i<x; i++)

{

if (x%i==0)

printf("%d ",i);

}

printf("\n");

}

int main(int argc, char\* argv[])

{

int x;

for (int i=1; i<argc; i++)

{

x= fork();

if(x==0)

{

printf("PID: %d Parent Process ID is: %d\nFactors of %d: ",getpid(),getppid(),atoi(argv[i]));

factor(atoi(argv[i]));

break;

}

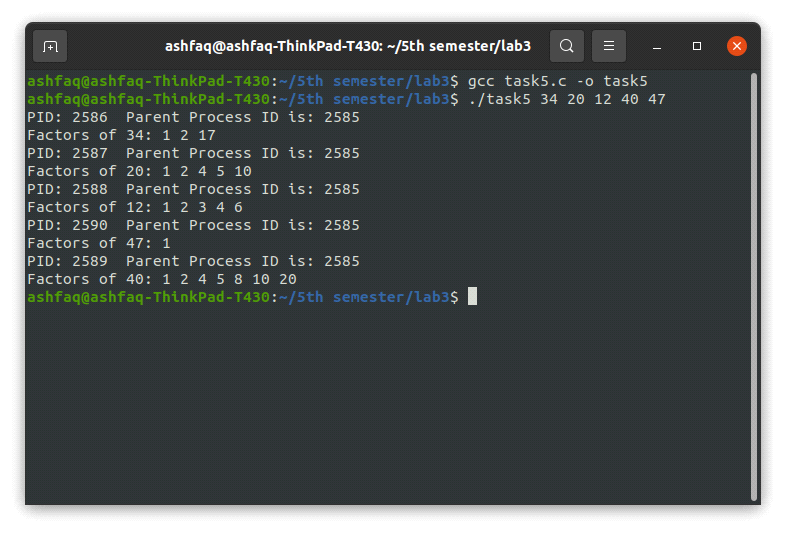
}

for (int i=1; i<argc; i++)

wait(NULL);

}

**Output:**



**2nd method:**

**Source Code:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

void factor(int x)

{

for(int i=1; i<x; i++)

{

if(x%i==0)

{

printf("%d ",i);

}

}

printf("\n");

}

int main()

{

int x;

printf("please! enter the size of array:");

scanf("%d",&x);

int arr[x];

printf("Please! Enter the array elements: \n");

for(int i=0; i<x; i++)

{

printf("arr[%d]=",i+1);

scanf("%d",&arr[i]);

}

for(int i=0; i<x; i++)

{

int z=fork();

if(z==0)

{

printf("PID: %d, PPID: %d, \nFactors of %d: ",getpid(),getppid(),arr[i]);

factor(arr[i]);

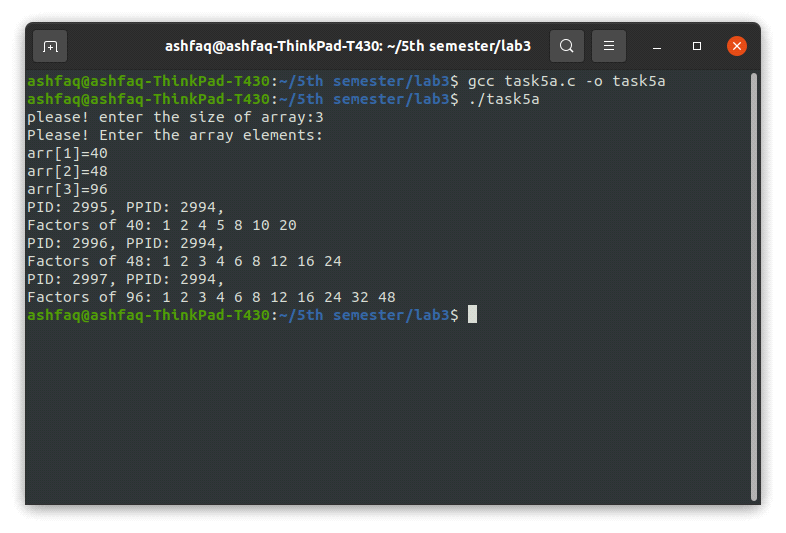
break;

}

}

};

**Output:**



**Task 6:** Write a program that creates an array of size 10,000. Initialize the array with random numbers. Create 10 child processes divide the array between them. Each child will add the portion and return their sum to parent process. Parent will add the results and display a final sum.

**Source code:**

**Using Wait and WEXITSTATUS:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

int main()

{

int arr[50];

for(int i=0;i<50;i++)

{

arr[i]=i;

}

int x,a=0,b=5,sum[10]={0},stat;

int total=0;

for (int i=0; i<10; i++)

{

x=fork();

if(x==0) //condition for process fan.

{

for(int j=a; j<b; j++)

{

sum[i]+=arr[j];

}

printf("PID: %d, PPID: %d, Return Sum Of child process %d: %d\n",getpid(),getppid(),i,sum[i]);

return sum[i];

break;

}

else if(x>0)

{

sleep(1);

a+=5;

b+=5;

wait(&stat);

total+=WEXITSTATUS(stat);

}

}

if(x>0)

{

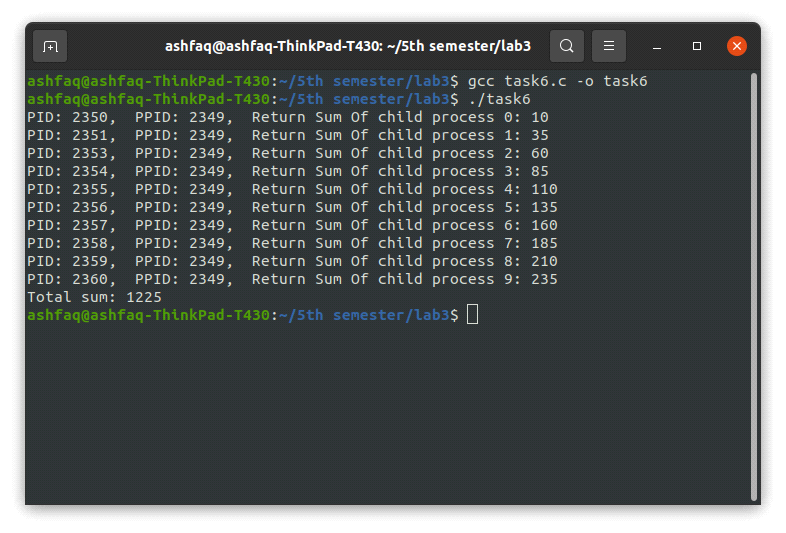
sleep(1);

printf("Total sum: %d\n",total);

}

}

**Output:**



THE END