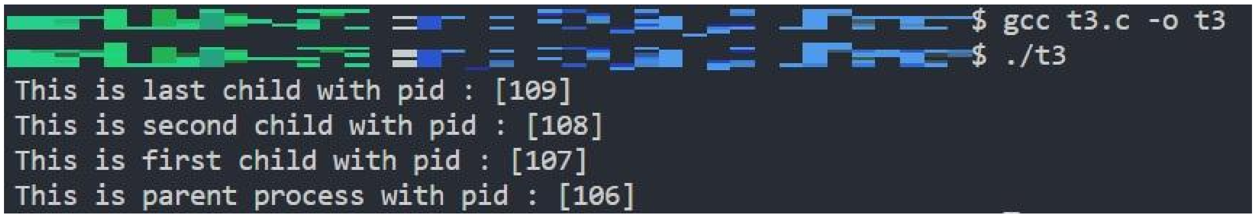
1. Write a program to create four processes(1 parent and 3 children) where they terminate in a sequence as follows:
    (a) Parent process terminates at last
    (b) First child terminates before parent and after second child
    (c) Second child terminates after last and before first child
    (d) Third child terminates first

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
    int f1 = fork(), f2 = fork();
    if (f1 > 0 && f2 > 0){
        sleep(15);
        printf("This is parent process with pid : [%d]\n", getpid());
    }else if(f1 == 0 && f2 > 0){
        sleep(10);
        printf("This is first child with pid : [%d]\n", getpid());
    }else if(f1 > 0 && f2 == 0){
        sleep(5);
        printf("This is second child with pid : [%d]\n", getpid());
    }else if(f1 == 0 && f2 == 0){
        printf("This is last child with pid : [%d]\n", getpid());
    }
}
```

OUTPUT:



```
[redacted]$ gcc t3.c -o t3
[redacted]$ ./t3
This is last child with pid : [109]
This is second child with pid : [108]
This is first child with pid : [107]
This is parent process with pid : [106]
```

2. Write a program to create a child process, which executes an already compiled factorial program.

CODE:

```bash
#!/bin/bash
echo -n "Enter the number to find factorial of : "
read number
myfact(){
if [[ $1 -le 1 ]]
then echo $1
else echo $(($1 * $(myfact $(( $1 - 1 )) )))
fi
}
echo -n "factorial is : "
myfact $number
```

3. Write a shell script to find “a” to the power “b” using a function.

CODE:

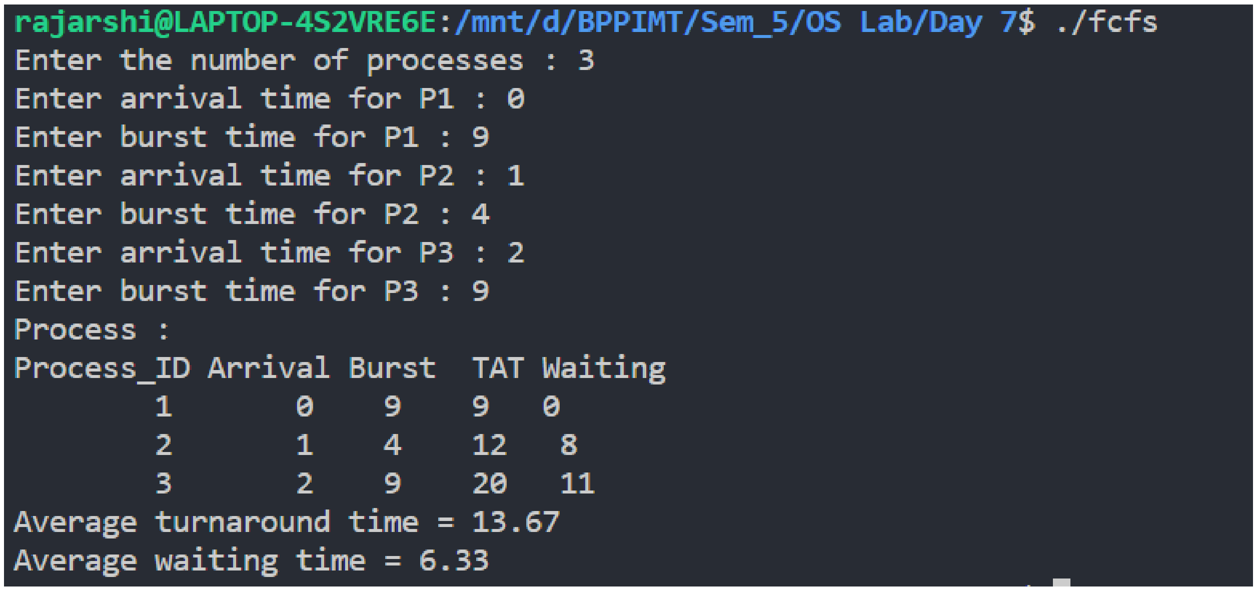
```
#!/bin/bash
# a and b are command line arguments
pow(){
num=$1
for((i=1; i<$2 ;i++))
do num=$(($num * $1))
done
echo $num
}
echo -n "$1 to the power $2 is : "
pow $1 $2
```

4. WAP to implement a FCFS CPU scheduling algorithm.

PID	Arrival Time	Burst Time
P1	0	9
P2	1	4
P3	2	9

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct p{
    int arrival, id, burst, waiting, tat;
};
int cmpfnc1(const void* a, const void* b){
    struct p* A = (struct p*)a;
    struct p* B = (struct p*)b;
    if (A-> arrival > B->arrival) return 1;
    if (B-> arrival > A->arrival) return -1;
    if (A->arrival == B->arrival) return A->id - B->id;
}
int cmpfnc2(const void* a, const void* b){
  struct p* A = (struct p*)a;
  struct p* B = (struct p*)b;
  return A->id - B->id;
}
void calc(struct p arr[],int n){
    qsort(arr, n, sizeof(struct p), cmpfnc1);
    int time = 0;
    float avg_tat = 0.0, avg_wait = 0.0;
    for(int i=0; i < n; i++){
        time+= arr[i].burst;
        arr[i].tat = time - arr[i].arrival;
        arr[i].waiting = arr[i].tat - arr[i].burst;
        avg_tat+= arr[i].tat, avg_wait+= arr[i].waiting;
    }
    printProcess(arr, n);
    printf("Average turnaround time = %.2f\n",avg_tat/n );
    printf("Average waiting time = %.2f\n",avg_wait/n );
}
void printProcess(struct p arr[],int n){
    qsort(arr, n, sizeof(struct p), cmpfnc2);
    printf("Process : \n");
    printf("Process_ID Arrival Burst  TAT Waiting\n");
    for(int i=0; i < n; i++) printf("\t%d\t%d    %d    %d   %d\n",arr[i].id,
arr[i].arrival, arr[i].burst, arr[i].tat, arr[i].waiting);
}
int main(){
    int n;
    printf("Enter the number of processes : ");
    scanf("%d",&n);
    struct p arr[n];
    for(int i=0; i < n; i++){
        printf("Enter arrival time for P%d : ", i+1);
        scanf("%d",&arr[i].arrival);
        printf("Enter burst time for P%d : ", i+1);
        scanf("%d",&arr[i].burst);
        arr[i].id = i + 1;
    }
    calc(arr, n);
    return 0;
}
```



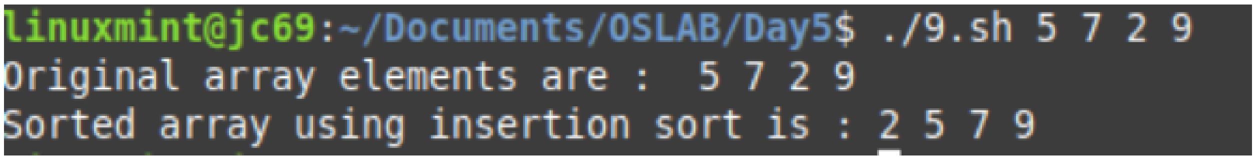
```
rajarshi@LAPTOP-4S2VRE6E:/mnt/d/BPPIMT/Sem_5/OS Lab/Day 7$ ./fcfs
Enter the number of processes : 3
Enter arrival time for P1 : 0
Enter burst time for P1 : 9
Enter arrival time for P2 : 1
Enter burst time for P2 : 4
Enter arrival time for P3 : 2
Enter burst time for P3 : 9
Process :
Process_ID Arrival Burst  TAT Waiting
        1       0    9    9   0
        2       1    4    12   8
        3       2    9    20   11
Average turnaround time = 13.67
Average waiting time = 6.33
```

5. Write a shell script for insertion sort.

CODE:

```
#!/bin/bash
arr=( "$@" )
echo "Original array elements are : " "${arr[*]}"
j=1
while [ $j -lt "$#" ]
do
    c=0
    k=$(expr $j - 1)
    while [ $k -ge 0 ]
    do
        if [ ${arr[k]} -gt ${arr[j]} ]
        then
            c=$(expr $c + 1)
        fi
    k=$(expr $k - 1)
    done

    x=$j
    y=$(expr $j - 1)

    while [ $c -gt 0 ]
    do
        # Swapping the elements
        temp=${arr[x]}
        arr[$x]=${arr[y]}
        arr[$y]=$temp

        x=$(expr $x - 1)
        y=$(expr $y - 1)
        c=$(expr $c - 1)
    done

    j=$(expr $j + 1)
done
printf "Sorted array using insertion sort is : "
echo "${arr[*]}"
```

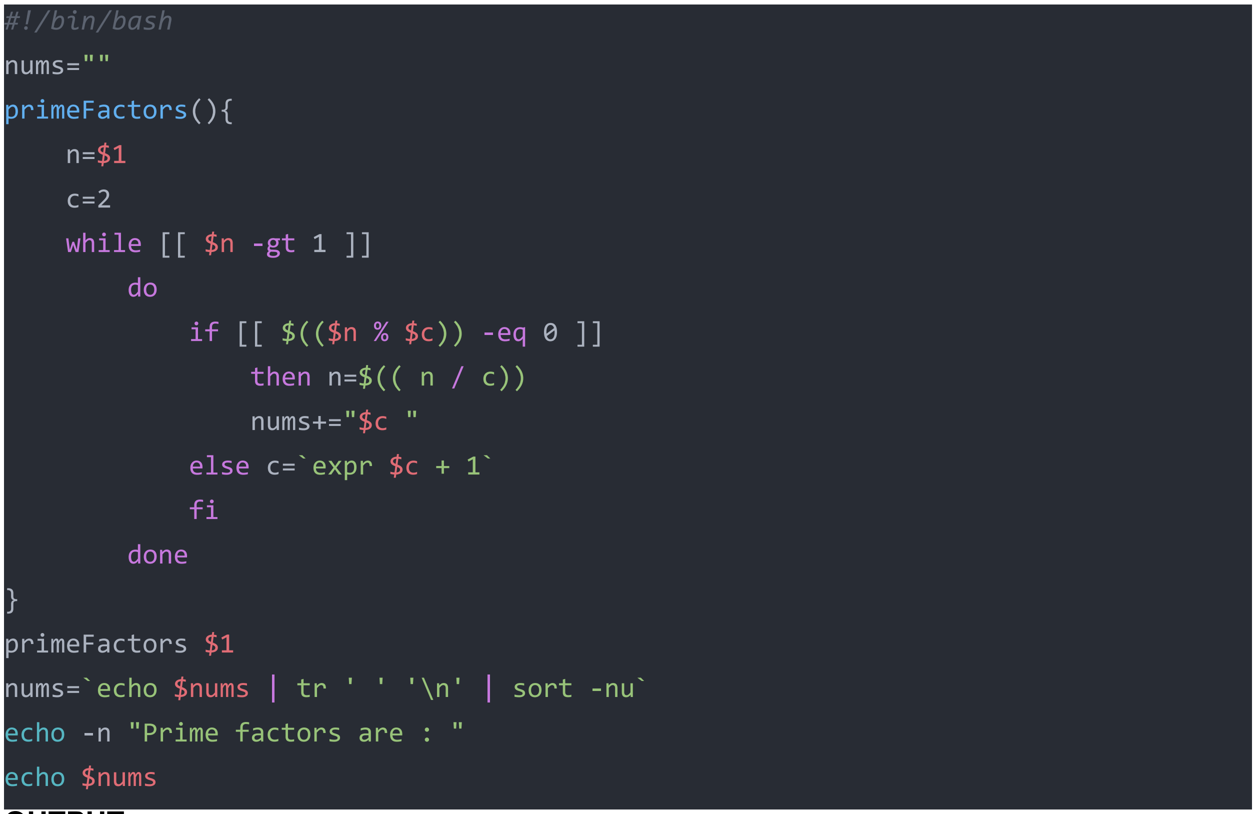
OUTPUT:



```
linuxmint@jc69:~/Documents/OSLAB/Day5$ ./9.sh 5 7 2 9
Original array elements are :  5 7 2 9
Sorted array using insertion sort is : 2 5 7 9
```

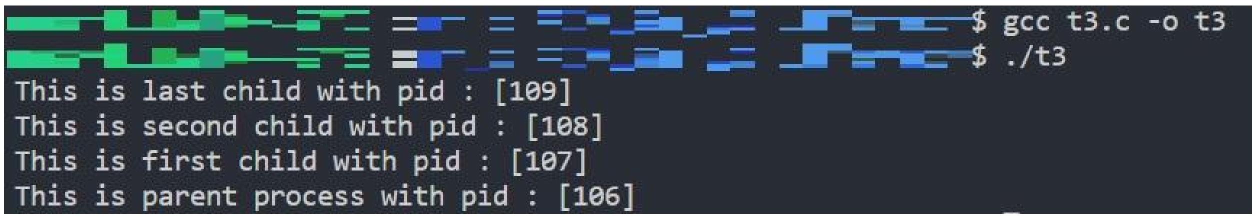
6. Write a shell script to find prime factors of a number.

CODE:



```
#!/bin/bash
nums=""
primeFactors(){
    n=$1
    c=2
    while [[ $n -gt 1 ]]
        do
            if [[ $(($n % $c)) -eq 0 ]]
                then n=$(( n / c))
                nums+="$c "
            else c=`expr $c + 1`
            fi
        done
}
primeFactors $1
nums=`echo $nums | tr ' ' '\n' | sort -nu`
echo -n "Prime factors are : "
echo $nums
```

OUTPUT:



```
[redacted]$ gcc t3.c -o t3
[redacted]$ ./t3
This is last child with pid : [109]
This is second child with pid : [108]
This is first child with pid : [107]
This is parent process with pid : [106]
```

7. WAP to show that the parent process identifier can never be zero of any child process and also explain the cause.

Ans : For the code part, demonstrate the orphan process.

In case of an orphan process, as its parent process has already executed and exited the process table, the orphan process(also in zombie mode) is left to run with init (having process ID 1 as its parent.

8. WAP to create multiple child processes and the child process may replace itself with another process while maintaining the same process ID.

// not possible logically

// ns_last_pid can be modified..

Checkout this code :

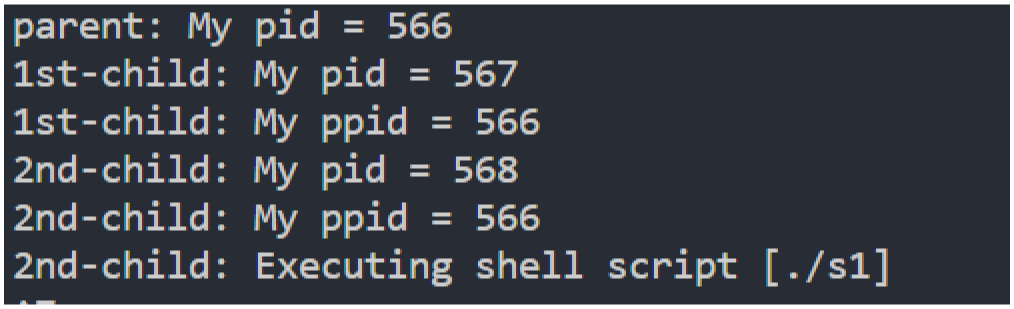
```
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int fd, pid;
    char buf[32];

    if (argc != 2)
     return 1;

    printf("Opening ns_last_pid...\n");
    fd = open("/proc/sys/kernel/ns_last_pid", O_RDWR | O_CREAT, 0644);
    if (fd < 0) {
        perror("Can't open ns_last_pid");
        return 1;
    }
    printf("Done\n");

    printf("Locking ns_last_pid...\n");
    if (flock(fd, LOCK_EX)) {
        close(fd);
        printf("Can't lock ns_last_pid\n");
        return 1;
    }
    printf("Done\n");

    pid = atoi(argv[1]);
    snprintf(buf, sizeof(buf), "%d", pid - 1);

    printf("Writing pid-1 to ns_last_pid...\n");
    if (write(fd, buf, strlen(buf)) != strlen(buf)) {
        printf("Can't write to buf\n");
        return 1;
    }
    printf("Done\n");
    printf("Forking...\n");
    int new_pid;
    new_pid = fork();
    if (new_pid == 0) {
        printf("I'm child!\n");
        exit(0);
    } else if (new_pid == pid) {
        printf("I'm parent. My child got right pid!\n");
    } else {
        printf("pid does not match expected one\n");
    }
    printf("Done\n");

    printf("Cleaning up...");
    if (flock(fd, LOCK_UN)) {
        printf("Can't unlock");
    }

    close(fd);

    printf("Done\n");

    return 0;
}
```

Link :
https://stackoverflow.com/questions/18122592/how-to-set-process-id-in-linux-for-a-specific-program

CODE according to SP:

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
int main(void)
{
pid_t xx;
int n;
char *argv[] = { "-d 1", NULL };
// argument list terminated by a NULL pointer. Read execve(2).
char *argp[] = { "TERM=linux", NULL };
// environment list terminated by a NULL pointer. Read execve(2).
printf( "parent: My pid = %u \n", getpid() );
// main(parent) tries to create a new process
```

```c
xx = fork();
if( xx == -1 ) { perror("fork-1"); exit(1); }
if ( xx == 0 ) // 1st-child process
{
printf("1st-child: My pid = %u \n", getpid() );
printf("1st-child: My ppid = %u \n", getppid() );
sleep(20);
printf("1st-child: Executing /usr/bin/top %s \n", argv[0]);
sleep(2);
n = execve ("/usr/bin/top", argv, argp);
if( n == -1 ) { perror("execve"); }
// [execve] never returns if successful
printf("[execve] ERROR!\n");
exit(2);
}
// parent process
// parent again tries to create a new process
xx = fork();
if( xx == -1 ) { perror("fork-2"); exit(3); }
if ( xx == 0 ) // 2nd-child process
{
printf( "2nd-child: My pid = %d \n",getpid() );
printf( "2nd-child: My ppid = %d \n",getppid() );
printf( "2nd-child: Executing shell script [./s1]\n" );
/*n = execl("./s1", NULL );
if( n == -1 ) { perror("execl"); }
// [execl] never returns if successful
printf("[execl ERROR !\n");
exit(4);*/
}
//parent process
wait(NULL);
wait(NULL);
return 0;
}
```
OUTPUT:



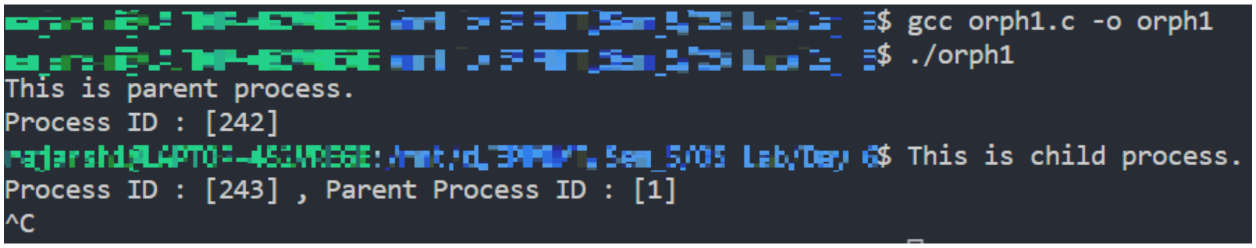
```
parent: My pid = 566
1st-child: My pid = 567
1st-child: My ppid = 566
2nd-child: My pid = 568
2nd-child: My ppid = 566
2nd-child: Executing shell script [./s1]
```

9. WAP to demonstrate an orphan process and also write the necessary conditions for zombie processes.

CODE:

```c
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

    pid_t child_id = fork();

    if (child_id > 0) {

        printf("This is parent process.\n");

        printf("Process ID : [%d]\n", getpid());

    }else if (child_id == 0) {

        //as child process sleeps for 20 seconds, parent process is executed
first

        //child will be executes with init as its parent process

        sleep(20);

        printf("This is child process.\n");

        printf("Process ID : [%d] , Parent Process ID : [%d]\n", getpid(),
getppid());

    }

    return 0;
}
```

OUTPUT:

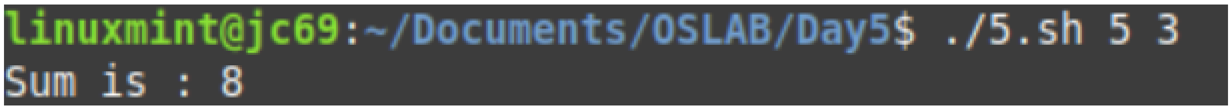


```
[redacted]$ gcc orph1.c -o orph1
[redacted]$ ./orph1
This is parent process.
Process ID : [242]
[redacted]$ This is child process.
Process ID : [243] , Parent Process ID : [1]
^C
```

10. Write a shell script to add two numbers using a function.

```
#!/bin/bash
add(){
      echo $(( $1 + $2 ))
      }
printf "Sum is : "
add $1 $2
```

OUTPUT:



```
linuxmint@jc69:~/Documents/OSLAB/Day5$ ./5.sh 5 3
Sum is : 8
```