OPERATING SYSTEMS

LAB ASESSMENT – 2

Name: **VIBHU KUMAR SINGH**
Reg. No: **19BCE0215**
Teacher: **Manikandan K.**

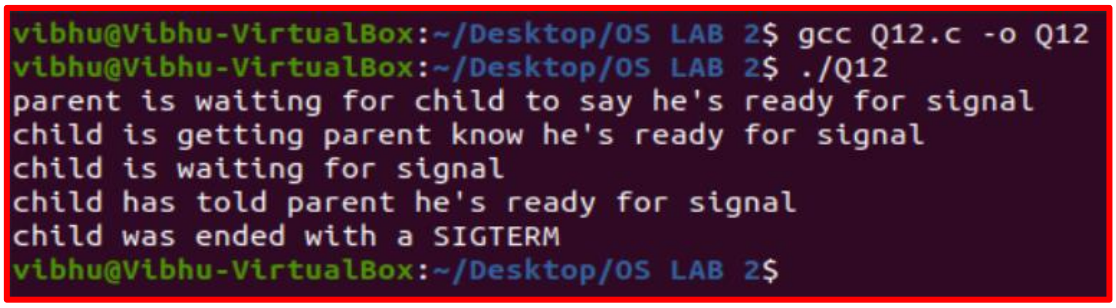
Q12) Write a C program to kill a process by specifying its name rather than its PID.

A12)

CODE:

```
#define _POSIX_SOURCE
#include<signal.h>
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>

int main()
{
        sigset_t sigset;
        int p[2], status;
        char c='z';
        pid_t pid;
        if (pipe(p) !=0)
                perror("pipe() error");
        else
        {
                if ((pid = fork()) == 0)
                {
                        sigemptyset(&sigset);
                        puts("child is getting parent know he's ready for signal");
                        write(p[1],&c,1);
                        puts("child is waiting for signal");
                        sigsuspend(&sigset);
                }
                puts("parent is waiting for child to say he's ready for signal");
                read(p[0],&c,1);
                puts("child has told parent he's ready for signal");
                kill(pid, SIGTERM);
                wait(&status);
                if (WIFSIGNALED(status))
                        if (WTERMSIG(status)==SIGTERM)
                                puts("child was ended with a SIGTERM");
                        else
                                printf("child vas ended with a %d signal\n", WTERMSIG(status));
                        else puts("child was not ended with a signal");
                        close(p[0]);
                        close(p[1]);
        }
}
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q12.c -o Q12
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q12
parent is waiting for child to say he's ready for signal
child is getting parent know he's ready for signal
child is waiting for signal
child has told parent he's ready for signal
child was ended with a SIGTERM
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

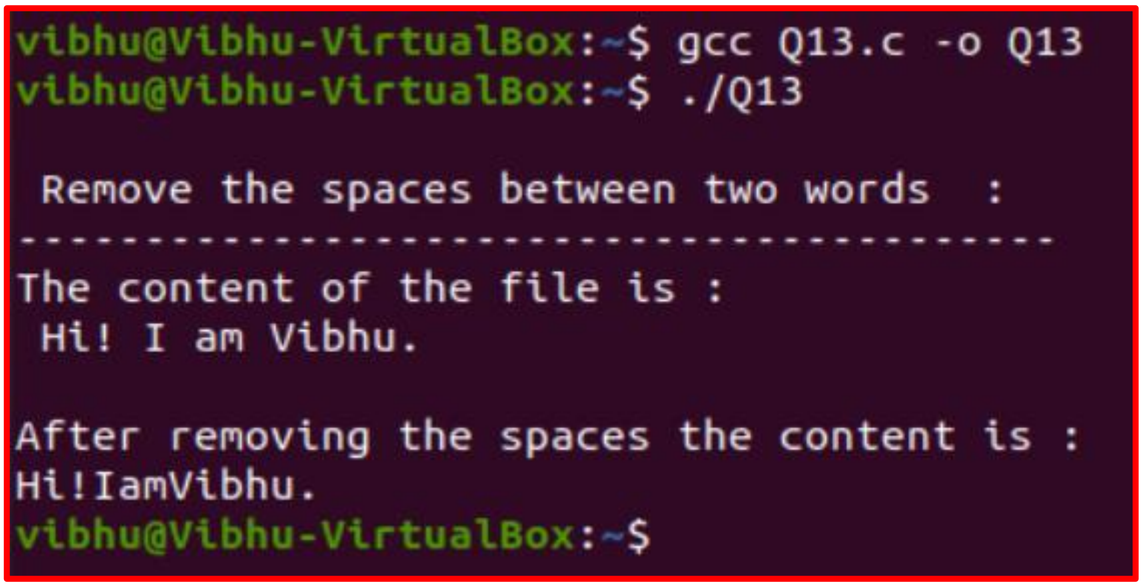
Q13) Create a file with few lines, Write a C program to read the file and delete the spaces more than one in the file (use UNIX file API's).

A13)

CODE:

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

int main()
{
        FILE * pfile;
        int a;
        printf("\n Remove the spaces between two words  :\n");
        printf("----------------------------------------\n");
        pfile=fopen ("file.txt","r");
        printf("The content of the file is :\n Hi! I am Vibhu.\n\n");
        printf("After removing the spaces the content is : \n");
        if (pfile)
        {
                do
                {
                a = fgetc (pfile);
                if (isgraph(a))
                        putchar (a);
                }while (a != EOF);
                fclose (pfile);
        }
        printf("\n");
        printf("Spaces Removed");
        return 0;
}
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~$ gcc Q13.c -o Q13
vibhu@Vibhu-VirtualBox:~$ ./Q13

 Remove the spaces between two words  :
-----------------------------------------
The content of the file is :
 Hi! I am Vibhu.

After removing the spaces the content is :
Hi!IamVibhu.
vibhu@Vibhu-VirtualBox:~$
```

Q14) Write a program

a. To create parent & child process and print their id.

b. To create a zombie process.

c. To create orphan process a).

A14)

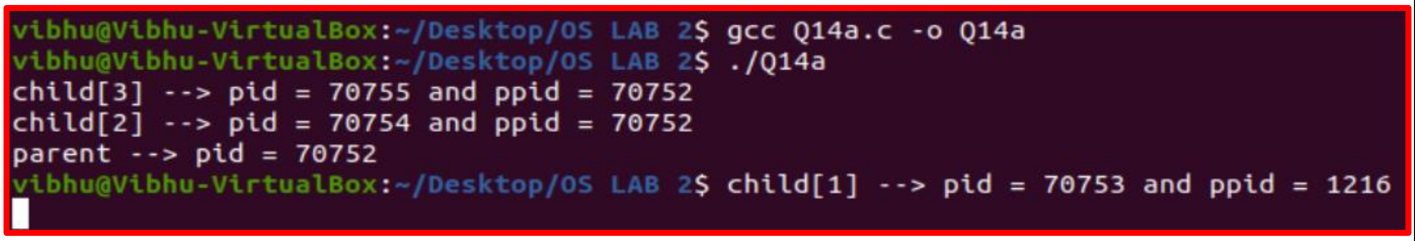
a)

CODE:

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

// Driver code
int main()
{
   int pid, pid1, pid2;

   // variable pid will store the
   // value returned from fork() system call
   pid = fork();

   // If fork() returns zero then it
   // means it is child process.
   if (pid == 0) {

      // First child needs to be printed
      // later hence this process is made
      // to sleep for 3 seconds.
      sleep(3);

      // This is first child process
      // getpid() gives the process
      // id and getppid() gives the
      // parent id of that process.
      printf("child[1] --> pid = %d and ppid = %d\n",
            getpid(), getppid());
   }

   else {
      pid1 = fork();
      if (pid1 == 0) {
         sleep(2);
         printf("child[2] --> pid = %d and ppid = %d\n",
               getpid(), getppid());
      }
      else {
         pid2 = fork();
         if (pid2 == 0) {
            // This is third child which is
            // needed to be printed first.
            printf("child[3] --> pid = %d and ppid = %d\n",
                  getpid(), getppid());
         }

         // If value returned from fork()
         // in not zero and >0 that means
```

```
            // this is parent process.
            else {
               // This is asked to be printed at last
               // hence made to sleep for 3 seconds.
               sleep(3);
               printf("parent --> pid = %d\n", getpid());
            }
         }
      }

      return 0;
   }
```

OUTPUT:



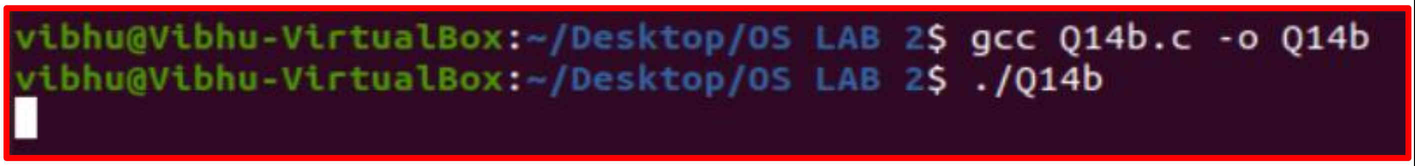
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q14a.c -o Q14a
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q14a
child[3] --> pid = 70755 and ppid = 70752
child[2] --> pid = 70754 and ppid = 70752
parent --> pid = 70752
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ child[1] --> pid = 70753 and ppid = 1216
```

b)

CODE:

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   // Fork returns process id
   // in parent process
   pid_t child_pid = fork();

   // Parent process
   if (child_pid > 0)
      sleep(20);

   // Child process
   else
      exit(0);

   return 0;
}
```

OUTPUT:



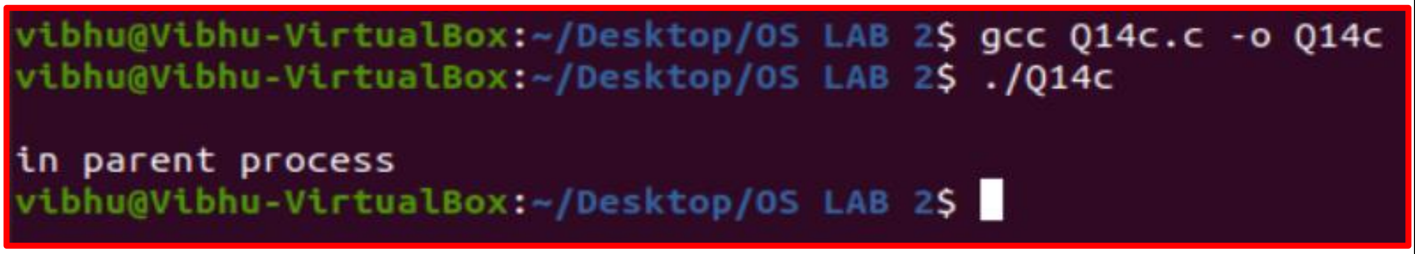
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q14b.c -o Q14b
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q14b
```

c)

CODE:

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

int main()
{
   // Create a child process
   int pid = fork();

   if (pid > 0)
      printf("in parent process\n");

   // Note that pid is 0 in child process
   // and negative if fork() fails
   else if (pid == 0)
   {
      sleep(30);
      printf("in child process");
   }

   return 0;
}
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q14c.c -o Q14c
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q14c

in parent process
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ 
```

Q15. Write a program

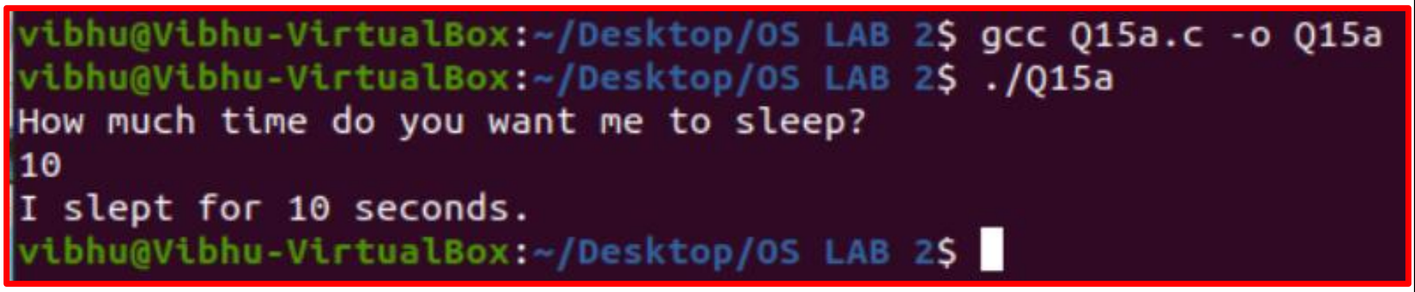
a. To make the process to sleep for few seconds.
b. To create a background process.
A15)

a)

CODE:

```
#include<stdlib.h>
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
void makeitsleep(int x)
{
        sleep(x);
}
int main()
{
        printf("How much time do you want me to sleep?\n");
        int n; scanf("%d",&n);
        makeitsleep(n);
        printf("I slept for %d seconds.\n",n);
        return(0);
}
```

OUTPUT:



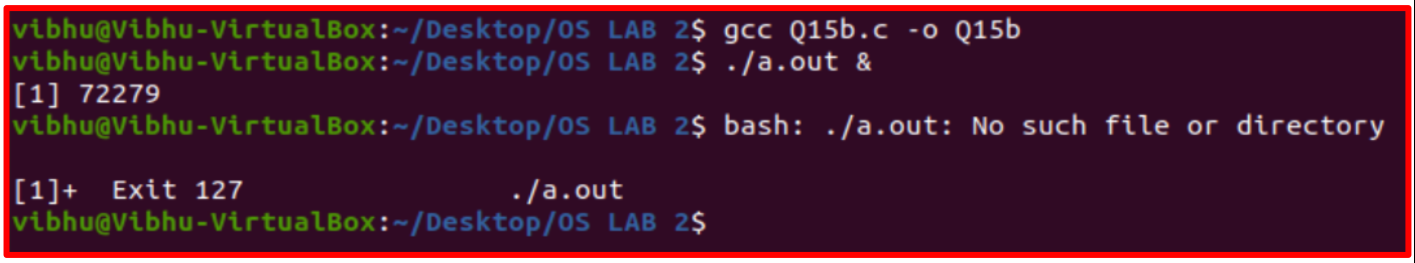
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q15a.c -o Q15a
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q15a
How much time do you want me to sleep?
10
I slept for 10 seconds.
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

b)

CODE:

```
#include<stdlib.h>
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
        printf("The process ID is: %d\n",getpid());
}
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q15b.c -o Q15b
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./a.out &
[1] 72279
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ bash: ./a.out: No such file or directory

[1]+  Exit 127                ./a.out
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

Q16) Write a program to create a thread and let the thread check whether the given number is prime or not.

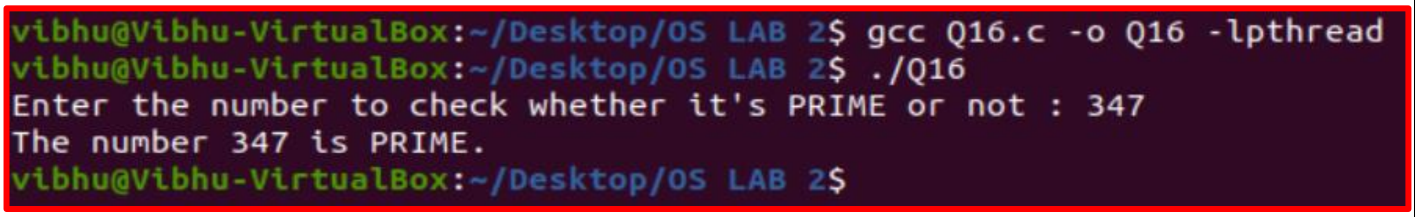
A16)

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *primeNumbers(void *vargp)
{
        int n, i, m = 0,flag = 0;
        printf("Enter the number to check whether it's PRIME or not : ");
        scanf("%d",&n);
        m = n/2;
        for(i = 2 ; i <= m ; i++)
        {
```

```
                    if(n%i == 0)
                    {
                            printf("The number %d is not PRIME.\n",n);
                            flag = 1;
                            break;
                    }
            }
            if(flag == 0)
            {
                    printf("The number %d is PRIME.\n",n);
            }
            return 0;
    }
    int main()
    {
            pthread_t thread_id;
            pthread_create(&thread_id, NULL, primeNumbers, NULL);
            pthread_join(thread_id, NULL);
            return 0;
    }
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q16.c -o Q16 -lpthread
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q16
Enter the number to check whether it's PRIME or not : 347
The number 347 is PRIME.
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

Q17) Design the following CPU Scheduling Algorithms to provide the performance analysis among them.

a. FCFS

b. PRIORITY

c. ROUND ROBIN

d. SJF FCFS

A17)

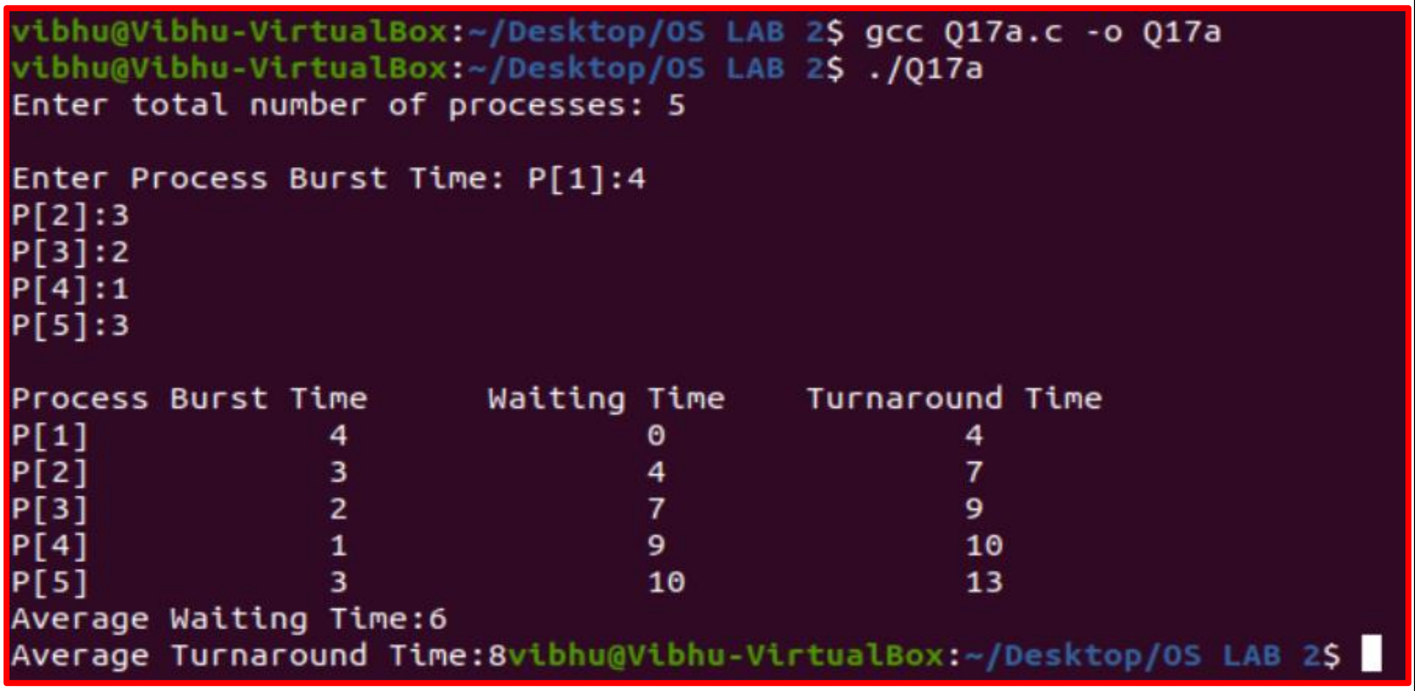
a)

CODE:

```
#include<stdio.h>
 int main()
{
   Int
n,burst_time[50],waiting_time[50],turnaround_time[50],avg_waiting_time=0,avg_turnaround_time
=0,i,j;
   printf("Enter total number of processes: ");
   scanf("%d",&n);
   printf("\nEnter Process Burst Time: ");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&burst_time[i]);
   }
```

```
    waiting_time[0]=0;
    for(i=1;i<n;i++)
    {
       waiting_time[i]=0;
       for(j=0;j<i;j++)
       waiting_time[i]+=burst_time[j];
    }
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
       turnaround_time[i]=burst_time[i]+waiting_time[i];
       avg_waiting_time+=waiting_time[i];
       avg_turnaround_time+=turnaround_time[i];
       printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,burst_time[i],waiting_time[i],turnaround_time[i]);
    }
    avg_waiting_time/=i;
    avg_turnaround_time/=i;
    printf("\nAverage Waiting Time:%d",avg_waiting_time);
    printf("\nAverage Turnaround Time:%d",avg_turnaround_time);
    return 0;
}
```

OUTPUT:



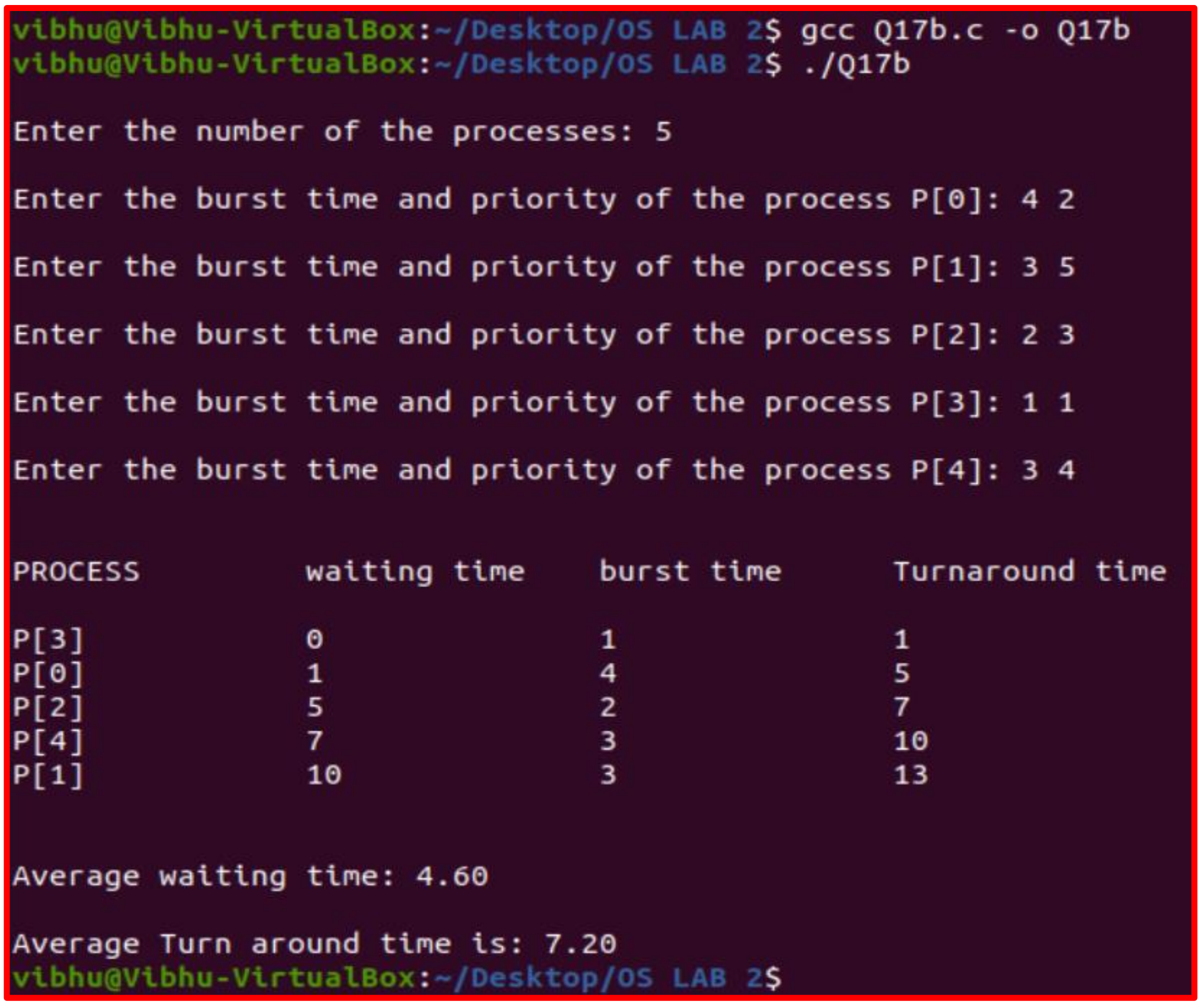
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q17a.c -o Q17a
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q17a
Enter total number of processes: 5

Enter Process Burst Time: P[1]:4
P[2]:3
P[3]:2
P[4]:1
P[5]:3

Process Burst Time      Waiting Time    Turnaround Time
P[1]            4               0               4
P[2]            3               4               7
P[3]            2               7               9
P[4]            1               9               10
P[5]            3               10              13
Average Waiting Time:6
Average Turnaround Time:8vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ 
```

b)

CODE:

```
#include <stdio.h>
int main()
{
    int bt[20],wt[20],p[20],tat[20],priority[20];
    float avwt=0,avtat=0;
    int i,j,n,temp,key;
    printf("\nEnter the number of the processes: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter the burst time and priority of the process P[%d]: ",i);
        scanf("%d",&bt[i]);
        scanf("%d",&priority[i]);
        p[i]=i;
    }
    for(i=0;i<n;i++)
    {
        key=i;
        for(j=i+1;j<n;j++)
        {
            if(priority[j]<priority[key])
            {
                key=j;
            }
        }
        temp=bt[i];
        bt[i]=bt[key];
        bt[key]=temp;
        temp=priority[i];
        priority[i]=priority[key];
        priority[key]=temp;
        temp=p[i];
        p[i]=p[key];
        p[key]=temp;
    }
    wt[0]=0;
    tat[0]=bt[0];
    avtat=tat[0];
    for(i=1;i<n;i++)
    {
            wt[i]=wt[i-1]+bt[i-1];
            tat[i]=tat[i-1]+bt[i];
            avwt+=wt[i];
            avtat+=tat[i];
    }
    avwt=avwt/n;
    avtat=avtat/n;
    printf("\n\nPROCESS\t\twaiting time\tburst time\tTurnaround time\n");
    printf("\n");
    for(i=0;i<n;i++)
    {
            printf("P[%d]\t\t%d\t\t%d\t\t%d\n",p[i],wt[i],bt[i],tat[i]);
    }
    printf("\n\nAverage waiting time: %.2f",avwt);
    printf("\n\nAverage Turn around time is: %.2f",avtat);
    printf("\n");
    return 0;
}
```

OUTPUT:



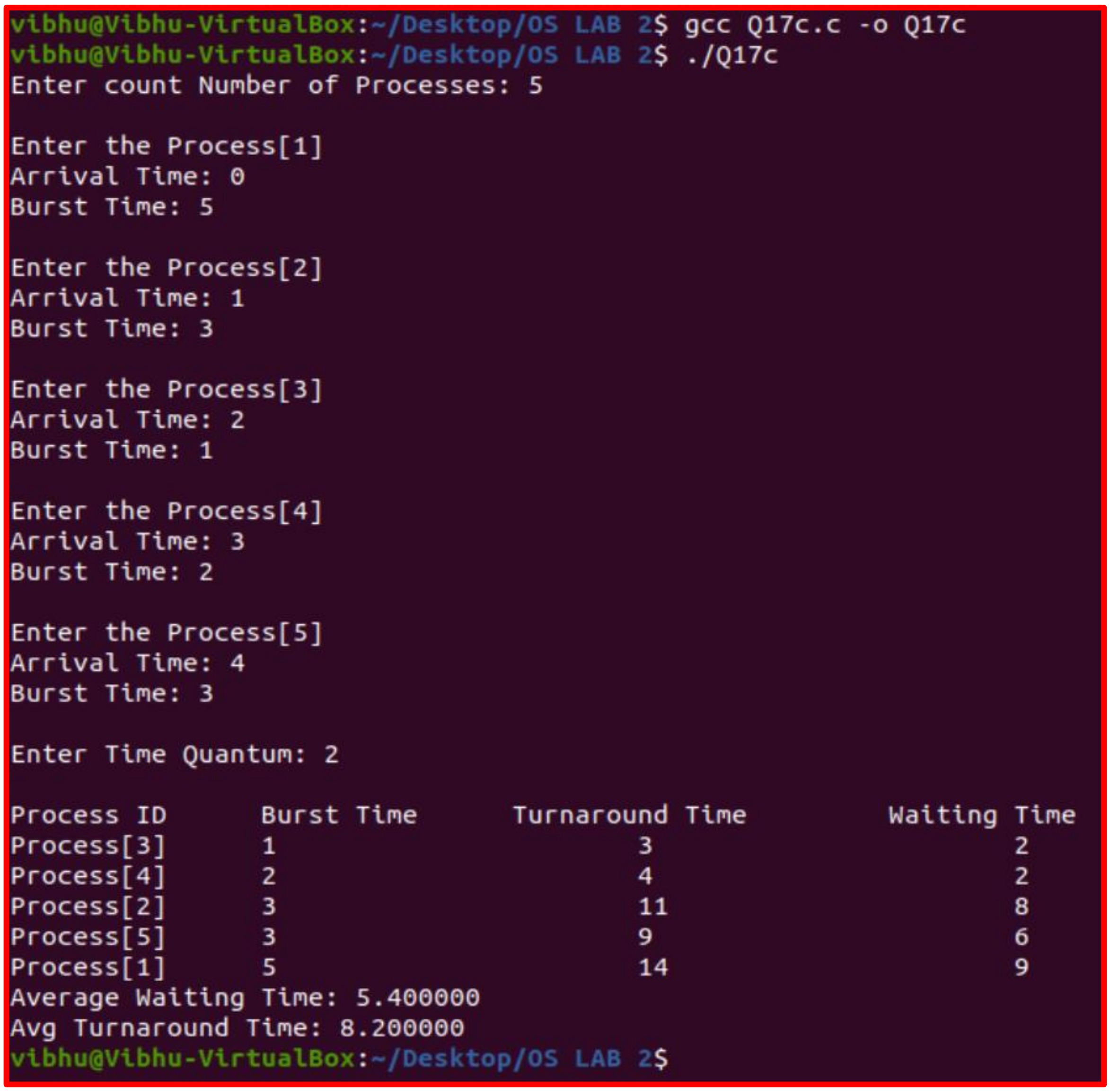
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q17b.c -o Q17b
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q17b

Enter the number of the processes: 5

Enter the burst time and priority of the process P[0]: 4 2

Enter the burst time and priority of the process P[1]: 3 5

Enter the burst time and priority of the process P[2]: 2 3

Enter the burst time and priority of the process P[3]: 1 1

Enter the burst time and priority of the process P[4]: 3 4


PROCESS         waiting time    burst time      Turnaround time

P[3]            0               1               1
P[0]            1               4               5
P[2]            5               2               7
P[4]            7               3               10
P[1]            10              3               13


Average waiting time: 4.60

Average Turn around time is: 7.20
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

c)

CODE:

```
#include<stdio.h>
int main()
{
    int limit,count = 0,a,i,count1 = 0,time_quantum;
    int waiting_time = 0, turnaround_time = 0, arrival_time[50], burst_time[50], temp[50];
    float average_waiting_time, average_turnaround_time;
    printf("Enter count Number of Processes: ");
    scanf("%d", &limit);
    a = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter the Process[%d]", i + 1);
        printf("\nArrival Time: ");
        scanf("%d", &arrival_time[i]);
        printf("\nBurst Time: ");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    printf("\nEnter Time Quantum: ");
    scanf("%d", &time_quantum);
    printf("\nProcess ID\tBurst Time\tTurnaround Time\t\tWaiting Time");
    for(count = 0, i = 0; a != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
        {
            count = count + temp[i];
            temp[i] = 0;
            count1 = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            count = count + time_quantum;
        }
        if(temp[i] == 0 && count1 == 1)
        {
            a--;
            printf("\nProcess[%d]\t%d\t\t\t%d\t\t\t%d", i + 1, burst_time[i], count - arrival_time[i],
count - arrival_time[i] - burst_time[i]);
            waiting_time = waiting_time + count - arrival_time[i] - burst_time[i];
            turnaround_time = turnaround_time + count - arrival_time[i];
            count1 = 0;
        }
        if(i == limit - 1)
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= count)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }
    average_waiting_time = waiting_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\nAverage Waiting Time: %f", average_waiting_time);
    printf("\nAvg Turnaround Time: %f", average_turnaround_time);
    return 0;
}
```

OUTPUT:



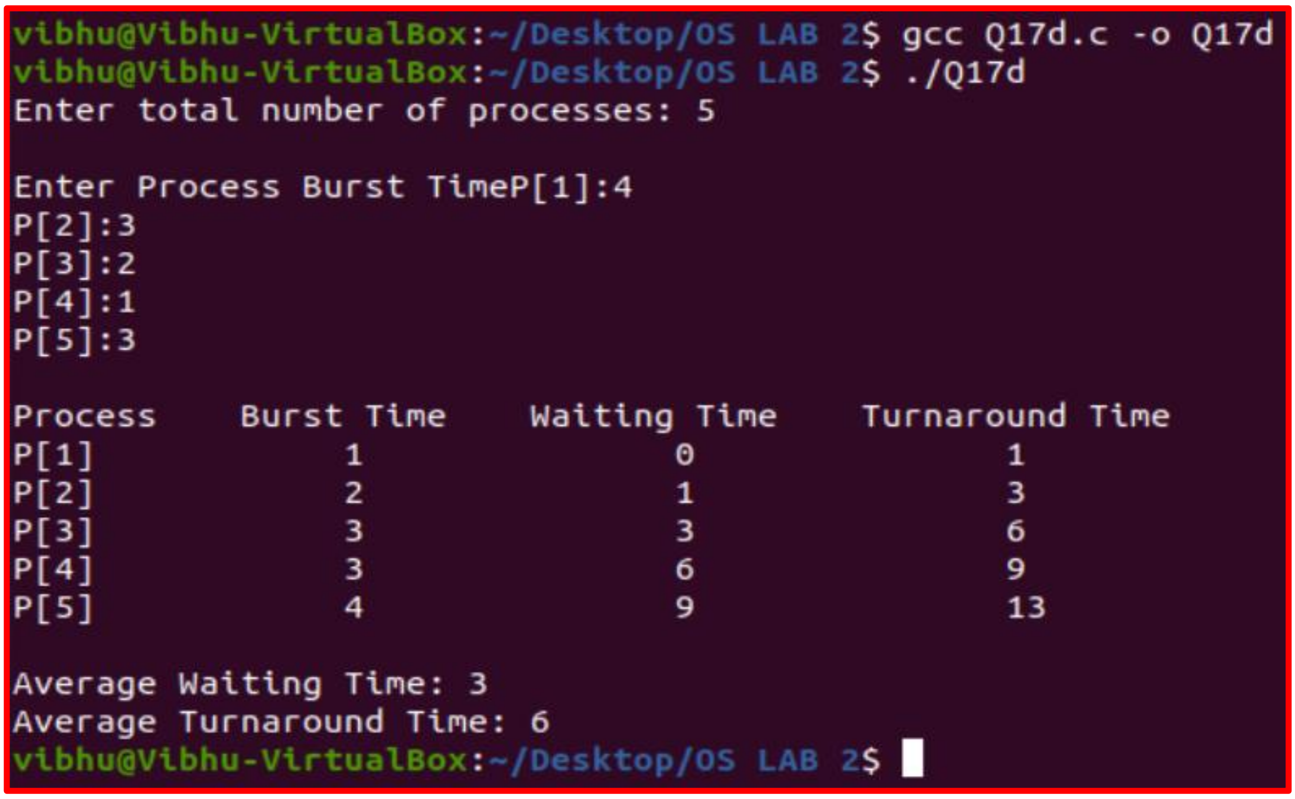
```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q17c.c -o Q17c
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q17c
Enter count Number of Processes: 5

Enter the Process[1]
Arrival Time: 0
Burst Time: 5

Enter the Process[2]
Arrival Time: 1
Burst Time: 3

Enter the Process[3]
Arrival Time: 2
Burst Time: 1

Enter the Process[4]
Arrival Time: 3
Burst Time: 2

Enter the Process[5]
Arrival Time: 4
Burst Time: 3

Enter Time Quantum: 2

Process ID      Burst Time      Turnaround Time         Waiting Time
Process[3]      1                    3                       2
Process[4]      2                    4                       2
Process[2]      3                    11                      8
Process[5]      3                    9                       6
Process[1]      5                    14                      9
Average Waiting Time: 5.400000
Avg Turnaround Time: 8.200000
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$
```

d)

CODE:

```
#include<stdio.h>
 int main()
{
   int
n,burst_time[50],waiting_time[50],turnaround_time[50],avg_waiting_time=0,avg_turnaround_time
=0,i,j,choice,temp,p[50];
   printf("Enter total number of processes: ");
   scanf("%d",&n);
   printf("\nEnter Process Burst Time");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&burst_time[i]);
      p[i]=i+1;
   }
   for(i=0;i<n;i++)
   {
      for(j=i+1;j<n;j++)
      {
         if(burst_time[j]<burst_time[i])
         {
                temp=burst_time[i];
                burst_time[i]=burst_time[j];
                burst_time[j]=temp;

                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
            }
        }
    }
    waiting_time[0]=0;
    for(i=1;i<n;i++)
    {
        waiting_time[i]=0;
        for(j=0;j<i;j++)
            waiting_time[i]+=burst_time[j];
    }
    printf("\nProcess    Burst Time    Waiting Time    Turnaround Time");
    for(i=0;i<n;i++)
    {
        turnaround_time[i]=burst_time[i]+waiting_time[i];
        avg_waiting_time+=waiting_time[i];
        avg_turnaround_time+=turnaround_time[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,burst_time[i],waiting_time[i],turnaround_time[i]);
    }
    avg_waiting_time/=i;
    avg_turnaround_time/=i;
    printf("\n\n Average Waiting Time:%d",avg_waiting_time);
    printf("\nAverage Turnaround Time:%d",avg_turnaround_time);
    return 0;
}
```

OUTPUT:



```
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ gcc Q17d.c -o Q17d
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ ./Q17d
Enter total number of processes: 5

Enter Process Burst TimeP[1]:4
P[2]:3
P[3]:2
P[4]:1
P[5]:3

Process    Burst Time    Waiting Time    Turnaround Time
P[1]            1               0               1
P[2]            2               1               3
P[3]            3               3               6
P[4]            3               6               9
P[5]            4               9               13

Average Waiting Time: 3
Average Turnaround Time: 6
vibhu@Vibhu-VirtualBox:~/Desktop/OS LAB 2$ 
```