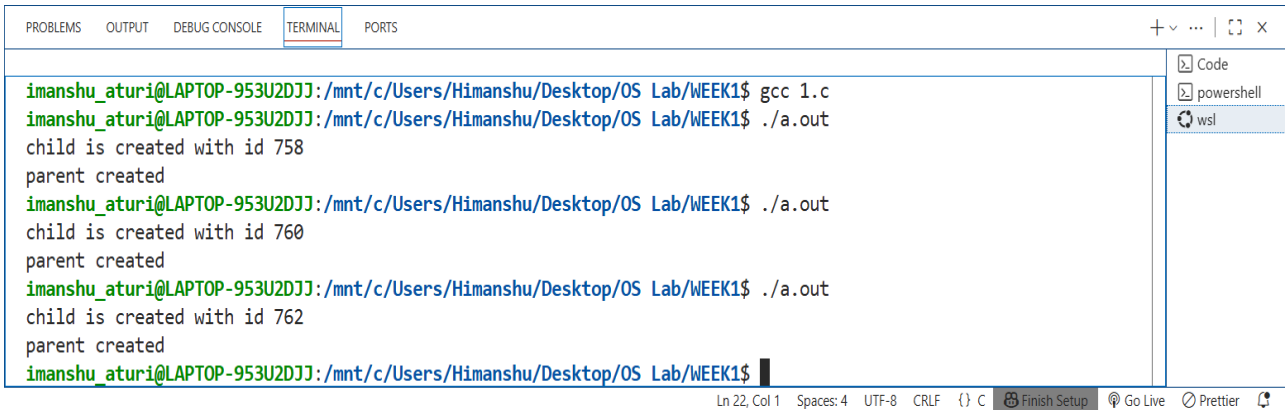
Week 1

Program 1) Write a program to create a child process using system call fork().

Source Code

```
#include <stdio.h>
#include <unistd.h>
int main()
{
   pid_t pid = fork();
   if (pid < 0)
   {
      printf("Fork failed!\n");
      return 1;
   }
   else if (pid == 0)
   {
      printf("Child Process PID: %d\n", getpid());
   }
   else
   {
      printf("Parent Process PID: %d, Child PID: %d\n", getpid(), pid);
   }
   return 0;
}
```

Output:

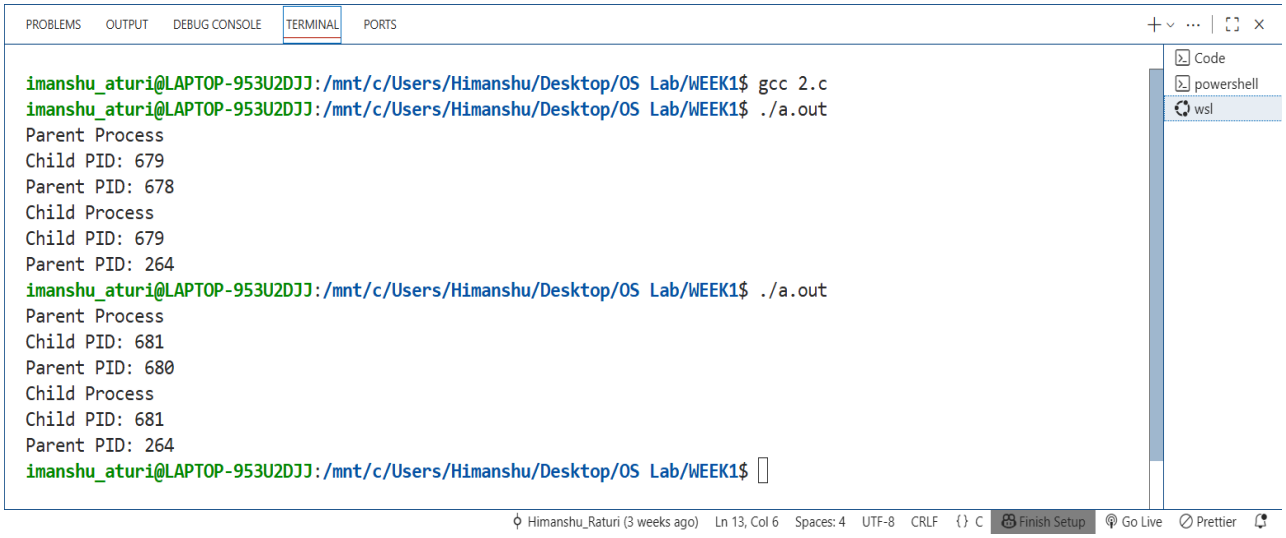

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code
powershell
wsl
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ gcc 1.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
child is created with id 758
parent created
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
child is created with id 760
parent created
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
child is created with id 762
parent created
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$
Ln 22, Col 1
Spaces: 4
UTF-8
CRLF
{} C
Finish Setup
Go Live
Prettier

Program 2) Write a program to print process Id's of parent and child process i.e. parent should print its own and its child process id while child process should print its own and its parent process id.

Source Code

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

int main()
{
   pid_t id = fork();
   if (id < 0)
   {
      printf("Forked Failed\n");
      return 1;
   }
   else if (id == 0)
   {
      printf("Child Process\n");
      printf("Child PID: %d\n", getpid());
      printf("Parent PID: %d\n", getppid());
   }
   else
   {
      printf("Parent Process\n");
      printf("Child PID: %d\n", id);
      printf("Parent PID: %d\n", getpid());
   }
   return 0;
}
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code
powershell
wsl
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ gcc 2.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
Parent Process
Child PID: 679
Parent PID: 678
Child Process
Child PID: 679
Parent PID: 264
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
Parent Process
Child PID: 681
Parent PID: 680
Child Process
Child PID: 681
Parent PID: 264
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$
Himanshu_Raturi (3 weeks ago)
Ln 13, Col 6
Spaces: 4
UTF-8
CRLF
{} C
Finish Setup
Go Live
Prettier

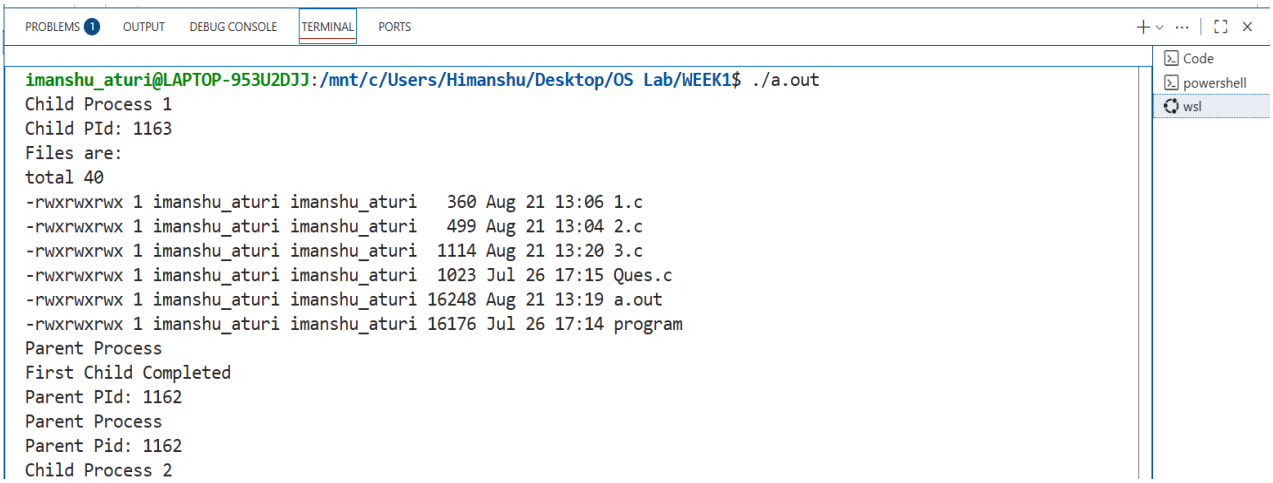
Program 3) Write a program to create child process. Make sure that parent process waits until child has not completed its execution. (use wait(), exit()) What will happen if parent process dies before child process? Illustrate it by creating one more child of parent process.

Source Code

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main()
{
   pid_t id1 = fork();
   if (id1 == -1)
   {
      printf("Fork Failed.\n");
      return 1;
   }
   else if (id1 == 0)
   {
      printf("Child Process 1\n");
      printf("Child PId: %d\n", getpid());
      printf("Files are:\n");
      execlp("ls", "ls", "-l", (char *)NULL);
      exit(0);
   }
   else if (id1 > 0)
   {
      wait(NULL);
      printf("Parent Process\n");
      printf("First Child Completed\n");
      printf("Parent PId: %d\n", getpid());
```

```
        pid_t id2 = fork();
        if (id2 < 0)
        {
            printf("Fork Failed\n");
            return 1;
        }
        else if (id2 == 0)
        {
            printf("Child Process 2\n");
            sleep(5);
            printf("Child PID: %d\n", getpid());
            exit(0);
        }
        else
        {
            printf("Parent Process\n");
            printf("Parent Pid: %d\n", getpid());
            exit(0);
        }
    }
    return 0;
}
```

OUTPUT


PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code
powershell
wsl
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
Child Process 1
Child PId: 1163
Files are:
total 40
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 360 Aug 21 13:06 1.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 499 Aug 21 13:04 2.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 1114 Aug 21 13:20 3.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 1023 Jul 26 17:15 Ques.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 16248 Aug 21 13:19 a.out
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 16176 Jul 26 17:14 program
Parent Process
First Child Completed
Parent PId: 1162
Parent Process
Parent Pid: 1162
Child Process 2

Program 4) Write a program to implement Orphan and Zombie Process.

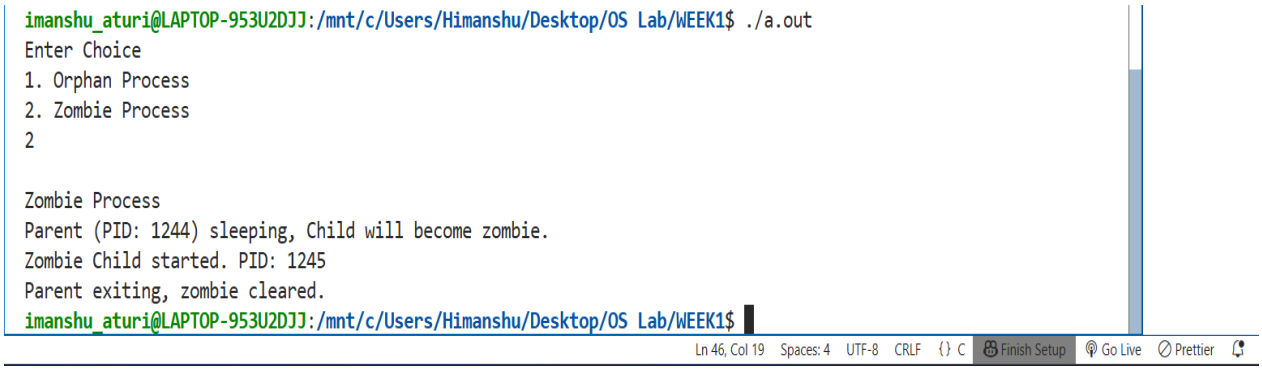
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
   int choice;
   printf("Enter Choice\n");
   printf("1. Orphan Process\n");
   printf("2. Zombie Process\n");
   scanf("%d", &choice);
   pid_t pid;
   if (choice == 1)
   {
      printf("\nOrphan Process\n");
      pid = fork();
      if (pid < 0)
      {
         printf("Fork failed");
         exit(1);
      }
      else if (pid == 0)
      {
         printf("Orphan Child started. PID: %d, Parent PID: %d\n", getpid(), getppid());
         sleep(5);
         printf("Orphan Child still running. PID: %d, New Parent PID: %d \n",
              getpid(), getppid());
         exit(0);
      }
```

```
        else
        {
            printf("Parent exiting, leaving child as orphan. PID: %d\n", getpid());
            exit(0);
        }
    }
    else if (choice == 2)
    {
        printf("\nZombie Process\n");
        pid = fork();
        if (pid < 0)
        {
            printf("Fork failed");
            exit(1);
        }
        else if (pid == 0)
        {
            printf("Zombie Child started. PID: %d\n", getpid());
            exit(0);
        }
        else
        {
            printf("Parent (PID: %d) sleeping, Child will become zombie.\n", getpid());
            sleep(20);
            printf("Parent exiting, zombie cleared.\n");
        }
    }
    return 0;
}
```

OUTPUT

```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ gcc 4.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ ./a.out
Enter Choice
1. Orphan Process
2. Zombie Process
1

Orphan Process
Parent exiting, leaving child as orphan. PID: 1213
Orphan Child started. PID: 1214, Parent PID: 1213
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK1$ Orphan Child still running. PID: 1214,
New Parent PID: 264
```



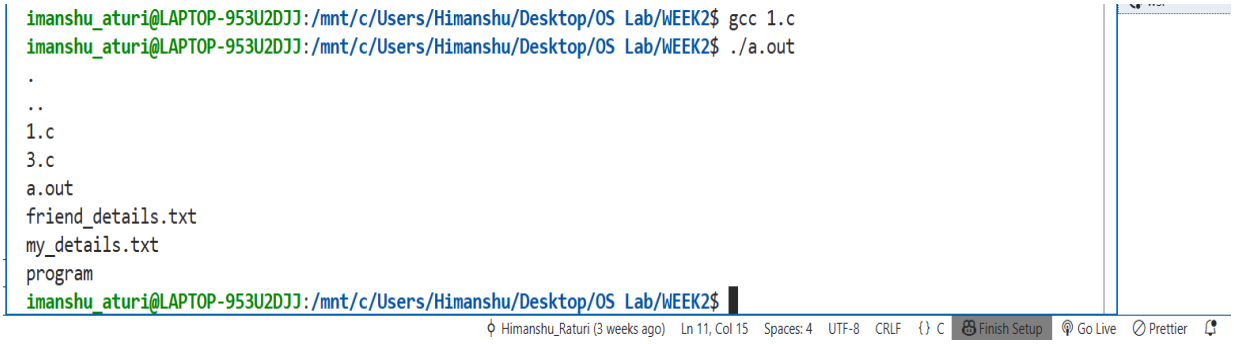
WEEK 2

Program 1) Write a program to open a directory and list its contents.

Source Code

```
#include <stdio.h>
#include <dirent.h>
int main()
{
   DIR *dir;
   struct dirent *entry;
   dir = opendir(".");
   if (dir == NULL)
   {
      printf("Error");
      return 1;
   }
   while ((entry = readdir(dir)) != NULL)
   {
      printf("%s\n", entry->d_name);
   }
   closedir(dir);
   return 0;
}
```

OUTPUT

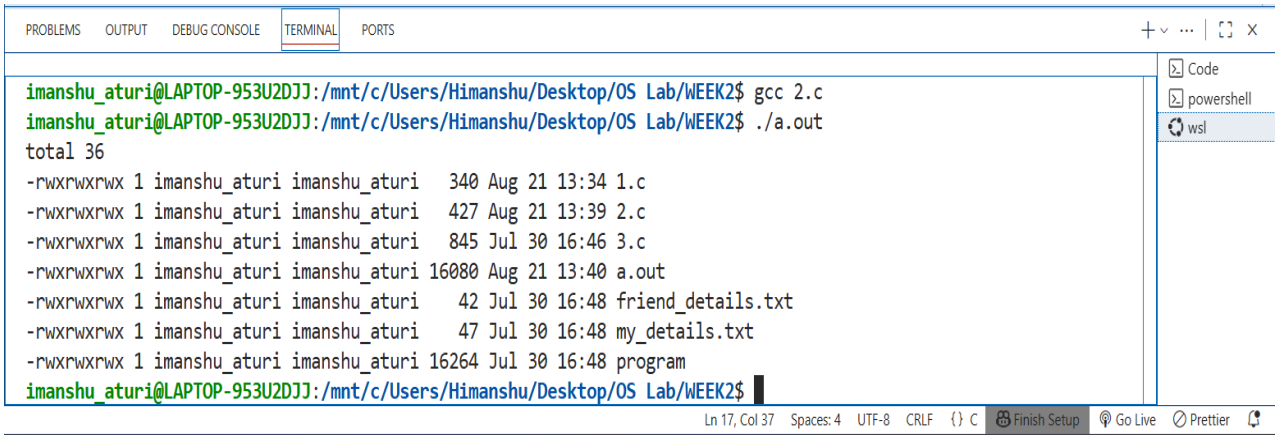

imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ gcc 1.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ ./a.out
.
..
1.c
3.c
a.out
friend_details.txt
my_details.txt
program
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$
Himanshu_Raturi (3 weeks ago)
Ln 11, Col 15
Spaces: 4
UTF-8
CRLF
C
Finish Setup
Go Live
Prettier

Program 2) Write a program to show working of execlp() system call by executing ls command.

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
   if (execlp("ls", "ls", "-l", (char *)NULL) == -1)
   {
      printf("execlp failed");
      exit(1);
   }
   printf("After execlp() call\n");
   return 0;
}
```

OUTPUT

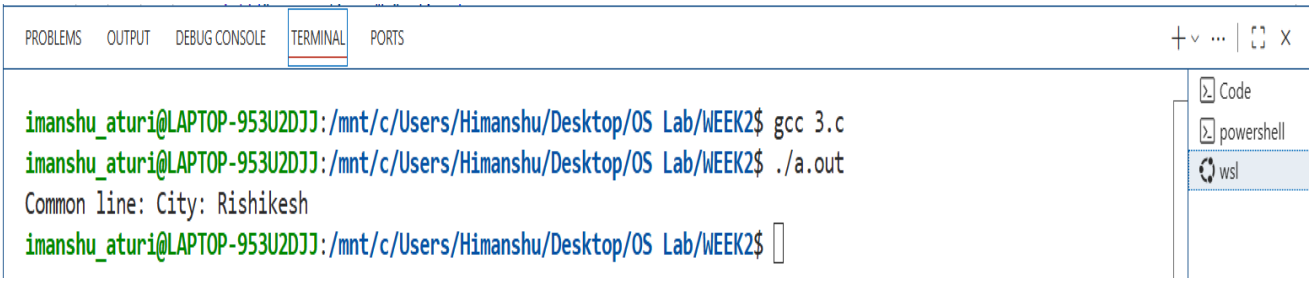

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code
powershell
wsl
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ gcc 2.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ ./a.out
total 36
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 340 Aug 21 13:34 1.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 427 Aug 21 13:39 2.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 845 Jul 30 16:46 3.c
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 16080 Aug 21 13:40 a.out
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 42 Jul 30 16:48 friend_details.txt
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 47 Jul 30 16:48 my_details.txt
-rwxrwxrwx 1 imanshu_aturi imanshu_aturi 16264 Jul 30 16:48 program
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$
Ln 17, Col 37
Spaces: 4
UTF-8
CRLF
{} C
Finish Setup
Go Live
Prettier

Program 3) Write a program to read a file and store your details in that file. Your program should also create one more file and store your friends details in that file. Once both files are created, print lines which are matching in both files.

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void  main()
{
   FILE *file1, *file2;
   char line1[100], line2[100];
   file1 = fopen("my_details.txt", "w");
   fprintf(file1, "Name: Himanshu Raturi\nRoll: 31\nCity: Rishikesh\n");
   fclose(file1);
   file2 = fopen("friend_details.txt", "w");
   fprintf(file2, "Name: Akhil Bhatt\nRoll: 30\nCity: Rishikesh\n");
   fclose(file2);
   file1 = fopen("my_details.txt", "r");
   file2 = fopen("friend_details.txt", "r");
   while (fgets(line1, sizeof(line1), file1))
   {
      rewind(file2);
      while (fgets(line2, sizeof(line2), file2))
      {
         if (strcmp(line1, line2) == 0)
            printf("Common line: %s", line1);
      }
   }
   fclose(file1);
   fclose(file2);
}
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ gcc 3.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$ ./a.out
Common line: City: Rishikesh
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK2$
Code
powershell
wsl


WEEK2 > friend_details.txt
1 Name: Akhil Bhatt
2 Roll: 30
3 City: Rishikesh
4


WEEK2 > my_details.txt
1 Name: Himanshu Raturi
2 Roll: 31
3 City: Rishikesh
4

WEEK 3

Program 1) Write a C program to implement FCFS algorithm.

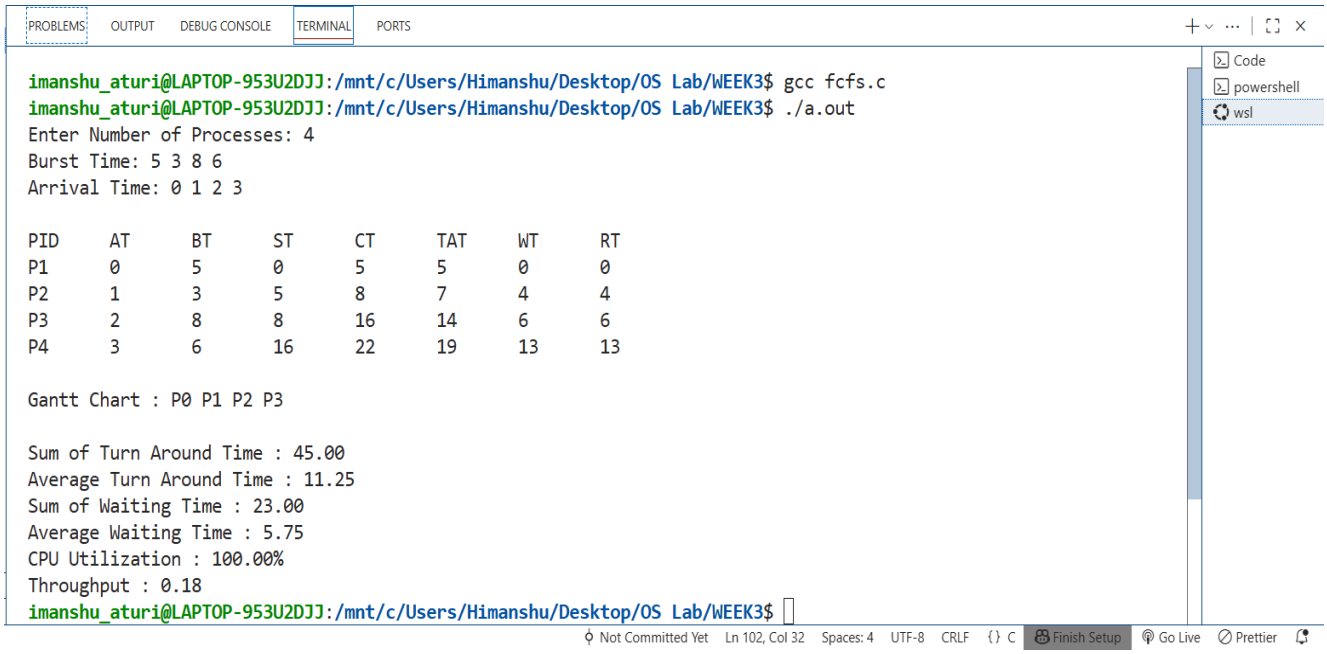
Source Code:

```
#include <stdio.h>

#include <stdlib.h>

typedef struct process

{

   int pid, arrival_time, burst_time, complete_time, int start_time, response_time, turn_around_time
, waiting_time;

} process;

int compare(const void *p1, const void *p2)

{

   int a = ((process *)p1)->arrival_time;

   int b = ((process *)p2)->arrival_time;

   if (a < b)

      return -1;

   else if (a > b)

      return 1;

   else

      return 0;

}

int main()

{

   float swt = 0, stat = 0;

   int sbt = 0, n;

   float awt = 0, atat = 0, cu = 0 , throughput = 0;

   printf("Enter Number of Processes: ");

   scanf("%d", &n);

   process p[n];

   printf("Burst Time: ");

   for (int i = 0; i < n; i++)
```

```
{
    scanf("%d", &p[i].burst_time);
    p[i].pid = i + 1;
}
printf("Arrival Time: ");
for (int i = 0; i < n; i++)
    scanf("%d", &p[i].arrival_time);
qsort(p, n, sizeof(process), compare);
for (int i = 0; i < n; i++)
{
    if (i == 0)
    {
        p[i].start_time = p[i].arrival_time;
    }
    else
    {
        if (p[i - 1].complete_time > p[i].arrival_time)
            p[i].start_time = p[i - 1].complete_time;
        else
            p[i].start_time = p[i].arrival_time;
    }
    p[i].complete_time = p[i].burst_time + p[i].start_time;
    p[i].turn_around_time = p[i].complete_time - p[i].arrival_time;
    p[i].waiting_time = p[i].turn_around_time - p[i].burst_time;
    p[i].response_time = p[i].start_time - p[i].arrival_time;
    swt += p[i].waiting_time;
    sbt += p[i].burst_time;
    stat += p[i].turn_around_time;
}
awt = swt / n;
atat = stat / n;
```

```
    float max_ct = p[0].complete_time;
    for (int i = 1; i < n; i++)
    {
        if (p[i].complete_time > max_ct)
            max_ct = p[i].complete_time;
    }
    cu = ((float)sbt / max_ct) * 100;
    throughput = n / max_ct;
    printf("\nPID\tAT\tBT\tST\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
                p[i].pid, p[i].arrival_time, p[i].burst_time,
                p[i].start_time, p[i].complete_time,
                p[i].turn_around_time, p[i].waiting_time, p[i].response_time);
    }
    printf("\nGantt Chart : ");
    for (int i = 0; i < n; i++)
        printf("P%d ", p[i].pid - 1);
    printf("\n");
    printf("\nSum of Turn Around Time : %.2f\nAverage Turn Around Time : %.2f\n", stat, atat);
    printf("Sum of Waiting Time : %.2f\nAverage Waiting Time : %.2f\n", swt, awt);
    printf("CPU Utilization : %.2f%%\nThroughput : %.2f\n", cu, throughput);
    return 0;
}
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code
powershell
wsl
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$ gcc fcfs.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$ ./a.out
Enter Number of Processes: 4
Burst Time: 5 3 8 6
Arrival Time: 0 1 2 3
PID AT BT ST CT TAT WT RT
P1 0 5 0 5 5 0 0
P2 1 3 5 8 7 4 4
P3 2 8 8 16 14 6 6
P4 3 6 16 22 19 13 13
Gantt Chart : P0 P1 P2 P3
Sum of Turn Around Time : 45.00
Average Turn Around Time : 11.25
Sum of Waiting Time : 23.00
Average Waiting Time : 5.75
CPU Utilization : 100.00%
Throughput : 0.18
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$
Not Committed Yet
Ln 102, Col 32
Spaces: 4
UTF-8
CRLF
C
Finish Setup
Go Live
Prettier

Program 2) Write a C program to implement Shortest Job First Non-Preemptive algorithm.

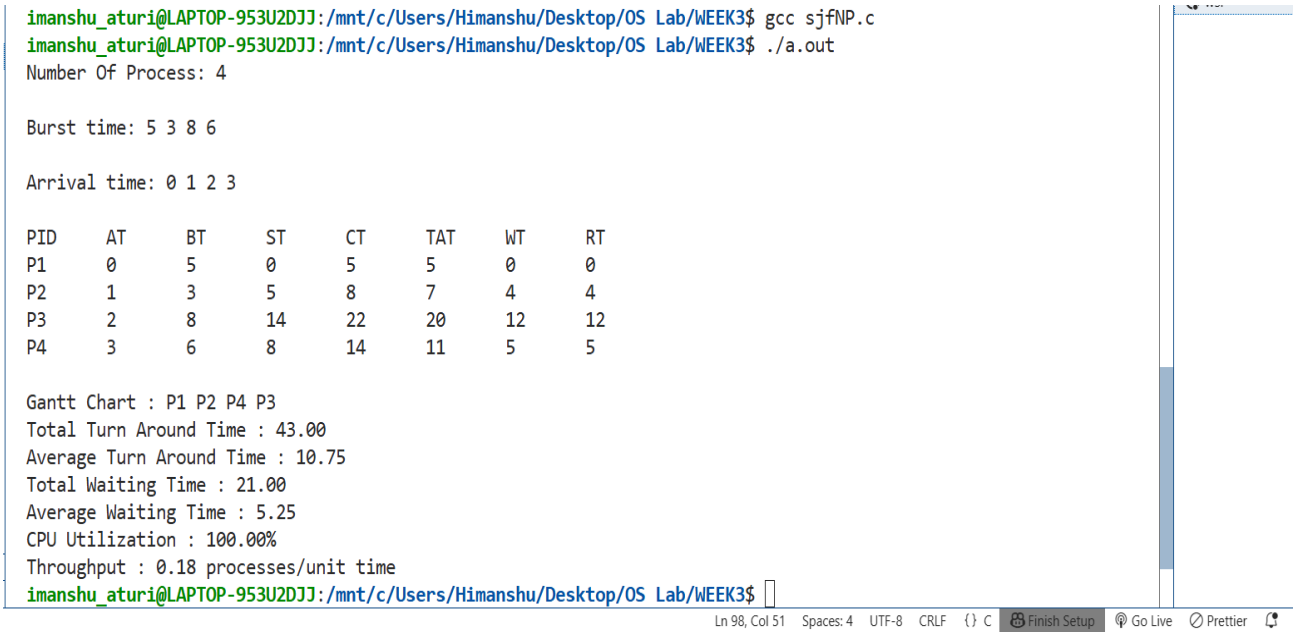
Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct process
{
   int pid, arr-time, burst_time, start_time, complete_time, turn_ard_time, wait_time,
response_time, is_completed;
} process;
void main()
{
   int n, sbt = 0;
   float swt = 0, stat = 0;
   float cu = 0, throughput = 0 , awt = 0 , atat = 0;
   printf("Number Of Process: ");
   scanf("%d", &n);
   process p[n];
   printf("\nBurst time: ");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &p[i].burst_time);
      p[i].pid = i + 1;
      p[i].is_completed = 0;
   }
   printf("\nArrival time: ");
   for (int i = 0; i < n; i++)
      scanf("%d", &p[i].arr_time);
   int completed = 0, curr_time = 0;
   float max_completion_time = 0;
   int gantt[n], g_ind = 0;
```

```
while (completed != n)
{
    int min_ind = -1;
    int min_bt = INT_MAX;
    for (int i = 0; i < n; i++)
    {
        if (p[i].arr_time <= curr_time && p[i].is_completed == 0)
        {
            if (p[i].burst_time < min_bt)
            {
                min_bt = p[i].burst_time;
                min_ind = i;
            }
            else if (p[i].burst_time == min_bt)
            {
                if (p[i].arr_time < p[min_ind].arr_time)
                    min_ind = i;
            }
        }
    }
    if (min_ind != -1)
    {
        p[min_ind].start_time = curr_time;
        p[min_ind].complete_time = p[min_ind].start_time + p[min_ind].burst_time;
        p[min_ind].turn_ard_time = p[min_ind].complete_time - p[min_ind].arr_time;
        p[min_ind].wait_time = p[min_ind].turn_ard_time - p[min_ind].burst_time;
        p[min_ind].response_time = p[min_ind].start_time - p[min_ind].arr_time;
        swt += p[min_ind].wait_time;
        stat += p[min_ind].turn_ard_time;
        sbt += p[min_ind].burst_time;
        if (p[min_ind].complete_time > max_completion_time)
```

```
            max_completion_time = p[min_ind].complete_time;
        gantt[g_ind++] = p[min_ind].pid;
        p[min_ind].is_completed = 1;
        completed++;
        curr_time = p[min_ind].complete_time;
      }
      else
      {
        curr_time++;
      }
    }
    awt = swt / n;
    atat = stat / n;
    cu = ((float)sbt / max_completion_time) * 100;
    throughput = (float)n / max_completion_time;
    printf("\nPID\tAT\tBT\tST\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++)
    {
      printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
            p[i].pid, p[i].arr_time, p[i].burst_time,
            p[i].start_time, p[i].complete_time,
            p[i].turn_ard_time, p[i].wait_time, p[i].response_time);
    }
    printf("\nGantt Chart : ");
    for (int i = 0; i < n; i++)
      printf("P%d ", gantt[i]); // if you want P0, P1... instead of P1, P2...
    printf("\nTotal Turn Around Time : %.2f\nAverage Turn Around Time : %.2f\n", stat, atat);
    printf("Total Waiting Time : %.2f\nAverage Waiting Time : %.2f\n", swt, awt);
    printf("CPU Utilization : %.2f%%\n", cu);
    printf("Throughput : %.2f processes/unit time\n", throughput);
}
```

OUTPUT


imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$ gcc sjfNP.c
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$ ./a.out
Number Of Process: 4
Burst time: 5 3 8 6
Arrival time: 0 1 2 3
PID AT BT ST CT TAT WT RT
P1 0 5 0 5 5 0 0
P2 1 3 5 8 7 4 4
P3 2 8 14 22 20 12 12
P4 3 6 8 14 11 5 5
Gantt Chart : P1 P2 P4 P3
Total Turn Around Time : 43.00
Average Turn Around Time : 10.75
Total Waiting Time : 21.00
Average Waiting Time : 5.25
CPU Utilization : 100.00%
Throughput : 0.18 processes/unit time
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK3$
Ln 98, Col 51 Spaces: 4 UTF-8 CRLF { } C Finish Setup Go Live Prettier

Program 3) Write a C program to implement Shortest Job First Preemptive algorithm.

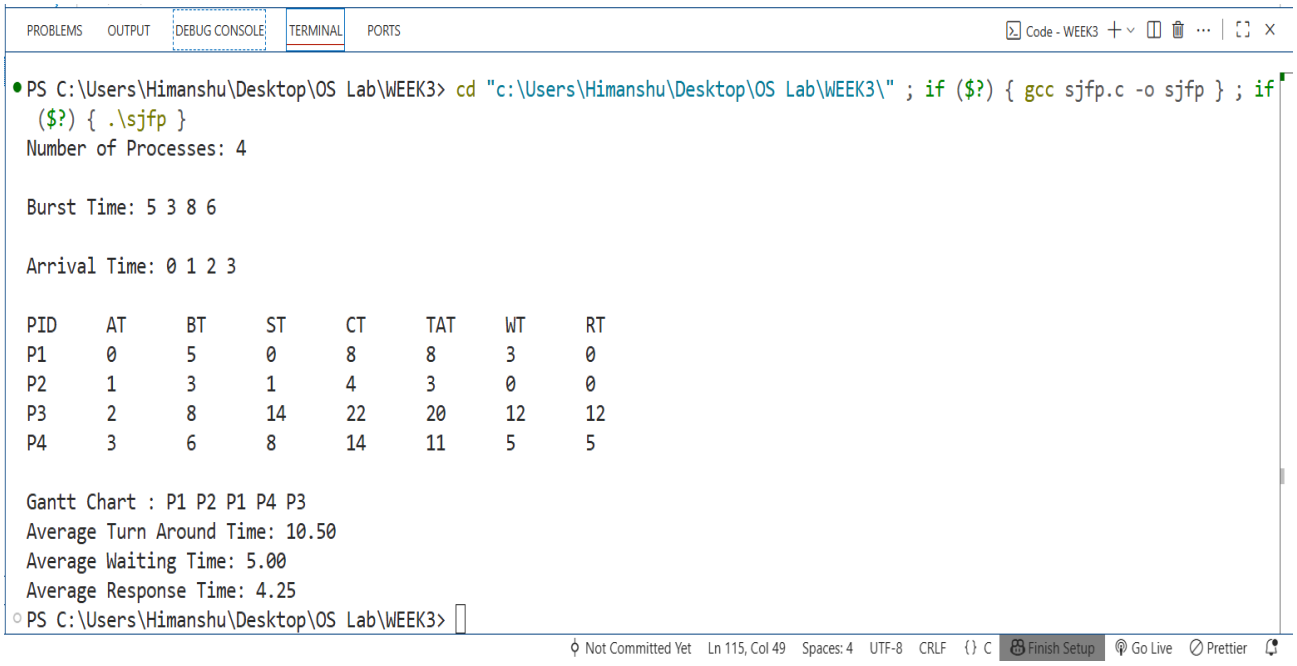
Source Code

```
#include <stdio.h>
#include <limits.h>
typedef struct process
{
   int pid, arr_time, burst_time, rem_time, start_time, complete_time, turn_ard_time,wait_time,
response_time , is_completed;
} process;
int main()
{
   int n;
   printf("Number of Processes: ");
   scanf("%d", &n);
   process p[n];
   int total_bt =0 , completed = 0, curr_time = 0;
   printf("\nBurst Time: ");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &p[i].burst_time);
      p[i].pid = i + 1;
      p[i].rem_time = p[i].burst_time; // initially remaining = burst
      p[i].is_completed = 0;
   }
   printf("\nArrival Time: ");
   for (int i = 0; i < n; i++)
      scanf("%d", &p[i].arr_time);
   float total_wt = 0, total_tat = 0, total_rt = 0;
   int gantt[100], g_index = 0;
   while (completed != n)
   {
```

```
int min_ind = -1;
int min_rt = INT_MAX;
for (int i = 0; i < n; i++)
{   if (p[i].arr_time <= curr_time && p[i].is_completed == 0)
    {
        if (p[i].rem_time < min_rt)
        {
            min_rt = p[i].rem_time;
            min_ind = i;
        }
        else if (p[i].rem_time == min_rt)
        {
            if (p[i].arr_time < p[min_ind].arr_time)
                min_ind = i;
        }
    }
}
if (min_ind != -1)
{
    if (p[min_ind].rem_time == p[min_ind].burst_time)
        p[min_ind].start_time = curr_time;
    gantt[g_index++] = p[min_ind].pid;
    p[min_ind].rem_time--;
    curr_time++;
    if (p[min_ind].rem_time == 0)
    {
        p[min_ind].complete_time = curr_time;
        p[min_ind].turn_ard_time = p[min_ind].complete_time - p[min_ind].arr_time;
        p[min_ind].wait_time = p[min_ind].turn_ard_time - p[min_ind].burst_time;
        p[min_ind].response_time = p[min_ind].start_time - p[min_ind].arr_time;
        total_wt += p[min_ind].wait_time;
```

```
            total_tat += p[min_ind].turn_ard_time;
            total_rt += p[min_ind].response_time;
            p[min_ind].is_completed = 1;
            completed++;
        }
    }
    else
    {
        curr_time++;
    }
}
printf("\nPID\tAT\tBT\tST\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++)
{
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
           p[i].pid, p[i].arr_time, p[i].burst_time,
           p[i].start_time, p[i].complete_time,
           p[i].turn_ard_time, p[i].wait_time, p[i].response_time);
}
printf("\nGantt Chart : ");
printf("P%d ", gantt[0]);
for (int i = 1; i < g_index; i++)
{
    if (gantt[i] != gantt[i - 1])
        printf("P%d ", gantt[i]);
}
printf("\nAverage Turn Around Time: %.2f", total_tat / n);
printf("\nAverage Waiting Time: %.2f", total_wt / n);
printf("\nAverage Response Time: %.2f\n", total_rt / n);
return 0;
}
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code - WEEK3
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK3> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK3\" ; if ($?) { gcc sjfp.c -o sjfp } ; if ($?) { .\sjfp }
Number of Processes: 4
Burst Time: 5 3 8 6
Arrival Time: 0 1 2 3
PID AT BT ST CT TAT WT RT
P1 0 5 0 8 8 3 0
P2 1 3 1 4 3 0 0
P3 2 8 14 22 20 12 12
P4 3 6 8 14 11 5 5
Gantt Chart : P1 P2 P1 P4 P3
Average Turn Around Time: 10.50
Average Waiting Time: 5.00
Average Response Time: 4.25
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK3>
Not Committed Yet
Ln 115, Col 49
Spaces: 4
UTF-8
CRLF
{} C
Finish Setup
Go Live
Prettier

WEEK 4

Program 1) Write a C program to implement Priority Scheduling(higher the number higher its priority).

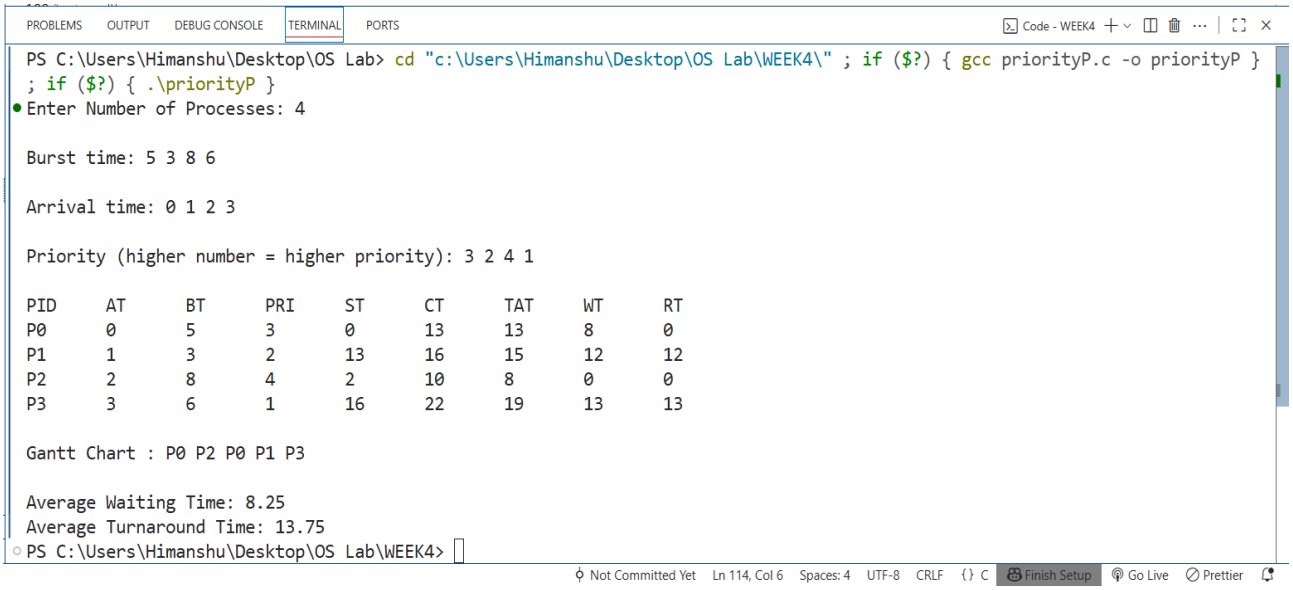
Source Code

```
#include <stdio.h>

#include <limits.h>

typedef struct process

{

   int pid, arr_time, burst_time, rem_time, priority, start_time, complete_time, turn_ard_time,
wait_time, response_time, is_completed;

} process;

void main()

{

   int n, completed = 0, curr_time = 0 , gantt[1000], g_index = 0;

   printf("Enter Number of Processes: ");

   scanf("%d", &n);

   process p[n];

   printf("\nBurst time: ");

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

   {

      scanf("%d", &p[i].burst_time);

      p[i].pid = i;

      p[i].rem_time = p[i].burst_time;

      p[i].is_completed = 0;

   }

   printf("\nArrival time: ");

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

      scanf("%d", &p[i].arr_time);

   printf("\nPriority (higher number = higher priority): ");

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

      scanf("%d", &p[i].priority);
```

```
float total_wt = 0, total_tat = 0, total_rt = 0;
while (completed != n)
{
    int idx = -1;
    int highest_priority = INT_MIN;
    for (int i = 0; i < n; i++)
    {
        if (p[i].arr_time <= curr_time && p[i].is_completed == 0)
        {
            if (p[i].priority > highest_priority)
            {
                highest_priority = p[i].priority;
                idx = i;
            }
            else if (p[i].priority == highest_priority)
            {
                if (p[i].arr_time < p[idx].arr_time)
                    idx = i;
            }
        }
    }
    if (idx != -1)
    {
        if (p[idx].rem_time == p[idx].burst_time)
            p[idx].start_time = curr_time; // first execution
        gantt[g_index++] = p[idx].pid;
        p[idx].rem_time--;
        curr_time++;
        if (p[idx].rem_time == 0)
        {
            p[idx].complete_time = curr_time;
```

```
            p[idx].turn_ard_time = p[idx].complete_time - p[idx].arr_time;
            p[idx].wait_time = p[idx].turn_ard_time - p[idx].burst_time;
            p[idx].response_time = p[idx].start_time - p[idx].arr_time;
            total_wt += p[idx].wait_time;
            total_tat += p[idx].turn_ard_time;
            total_rt += p[idx].response_time;
            p[idx].is_completed = 1;
            completed++;
        }
    }
    else
        curr_time++;
}
printf("\nPID\tAT\tBT\tPRI\tST\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++)
{
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
           p[i].pid, p[i].arr_time, p[i].burst_time, p[i].priority,
           p[i].start_time, p[i].complete_time,
           p[i].turn_ard_time, p[i].wait_time, p[i].response_time);
}
printf("\nGantt Chart : ");
printf("P%d ", gantt[0]);
for (int i = 1; i < g_index; i++)
{
    if (gantt[i] != gantt[i - 1])
        printf("P%d ", gantt[i]);
}
printf("\n\nAverage Waiting Time: %.2f", total_wt / n);
printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);
}
```

OUTPUT


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code - WEEK4
PS C:\Users\Himanshu\Desktop\OS Lab> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK4\" ; if ($?) { gcc priorityP.c -o priorityP } ; if ($?) { .\priorityP }
Enter Number of Processes: 4
Burst time: 5 3 8 6
Arrival time: 0 1 2 3
Priority (higher number = higher priority): 3 2 4 1
PID AT BT PRI ST CT TAT WT RT
P0 0 5 3 0 13 13 8 0
P1 1 3 2 13 16 15 12 12
P2 2 8 4 2 10 8 0 0
P3 3 6 1 16 22 19 13 13
Gantt Chart : P0 P2 P0 P1 P3
Average Waiting Time: 8.25
Average Turnaround Time: 13.75
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK4>
Not Committed Yet
Ln 114, Col 6
Spaces: 4
UTF-8
CRLF
C
Finish Setup
Go Live
Prettier