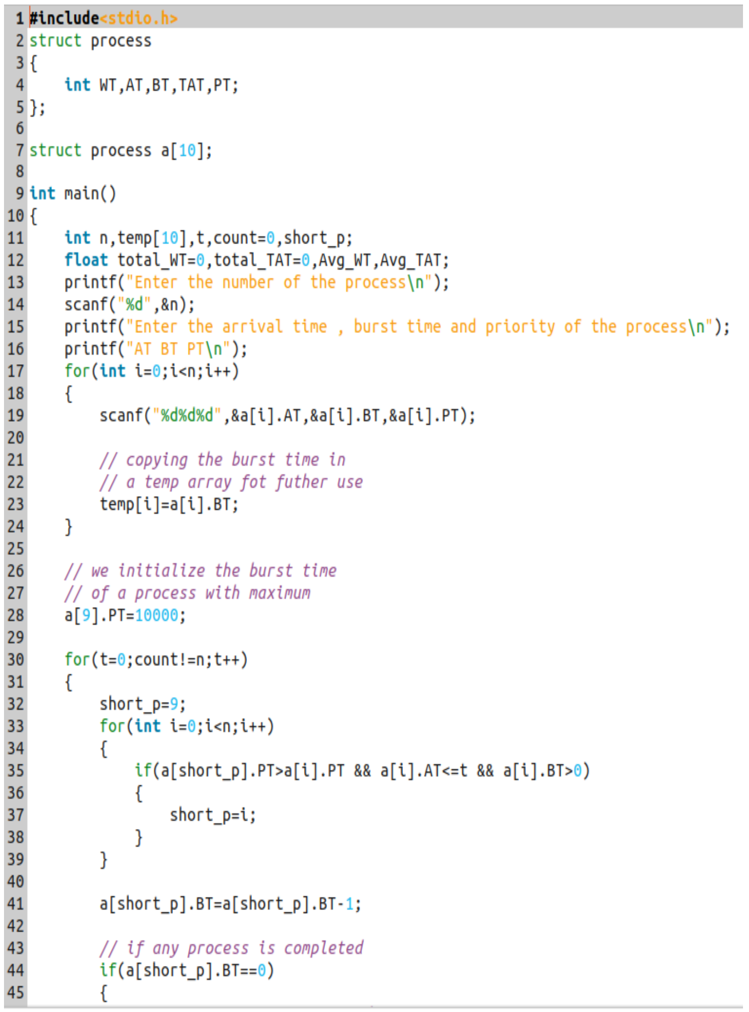
INDEX

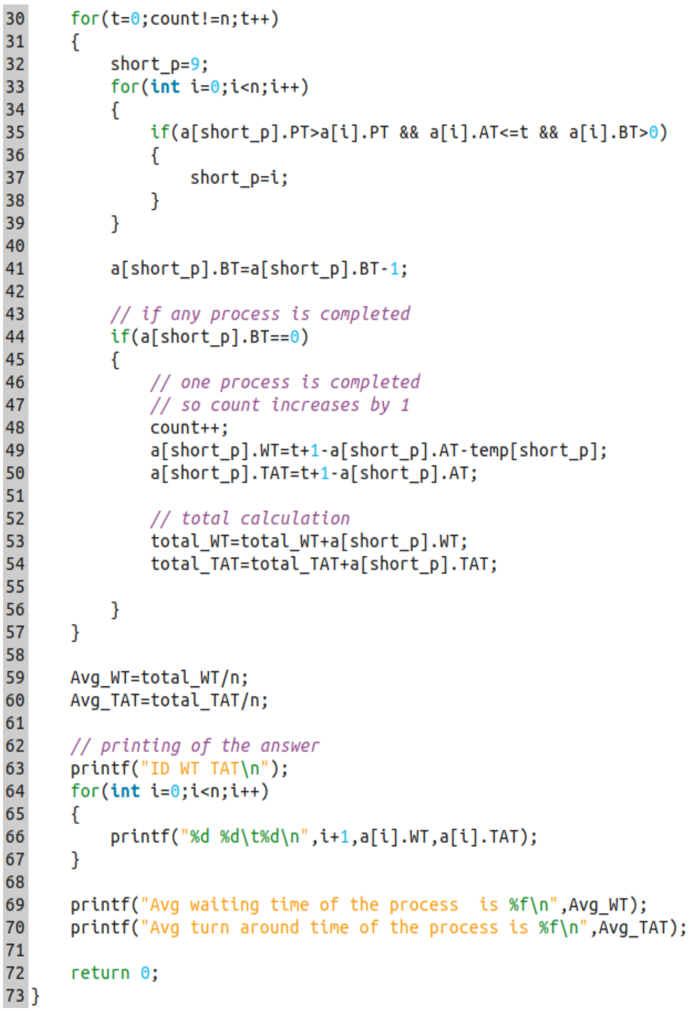
List of Experiments

1. Practice Linux shell commands.
2. Write C programs using fork(), getpid(), getppid() and exec() system calls.
3. Write a C program to represent a family of processes as a tree.
4. Write a program to simulate FCFS CPU scheduling algorithm.
5. Write a program to simulate SJF scheduling algorithm.
6. Write a program to simulate pre-emptive Priority scheduling algorithm.
7. Write a program to simulate Round Robin scheduling algorithm.
8. Write a program to simulate Multilevel Feedback Queue scheduling algorithm.
9. Write a program to simulate deadlock avoidance.
10. Write a program to simulate deadlock detection.
11. Write a program to simulate best-fit contiguous memory allocation.
12. Write a program to simulate FIFO page replacement algorithm.
13. Write a program to simulate LRU page replacement algorithm.
14. Write a program to simulate Second Chance page replacement algorithm. (Not to be done)
15. Write a program to simulate Enhanced Second Chance page replacement algorithm. (Not to be done)
16. Write a program to simulate LFU page replacement algorithm. (Not to be done)
17. Write a program to simulate FCFS disk scheduling algorithm.
18. Write a program to simulate SSTF disk scheduling algorithm.
19. Write a program to simulate C-SCAN disk scheduling algorithm.
20. Write a program to simulate LOOK disk scheduling algorithm.

EXPERIMENT-6 (PREEMPTIVE PRIORITY)

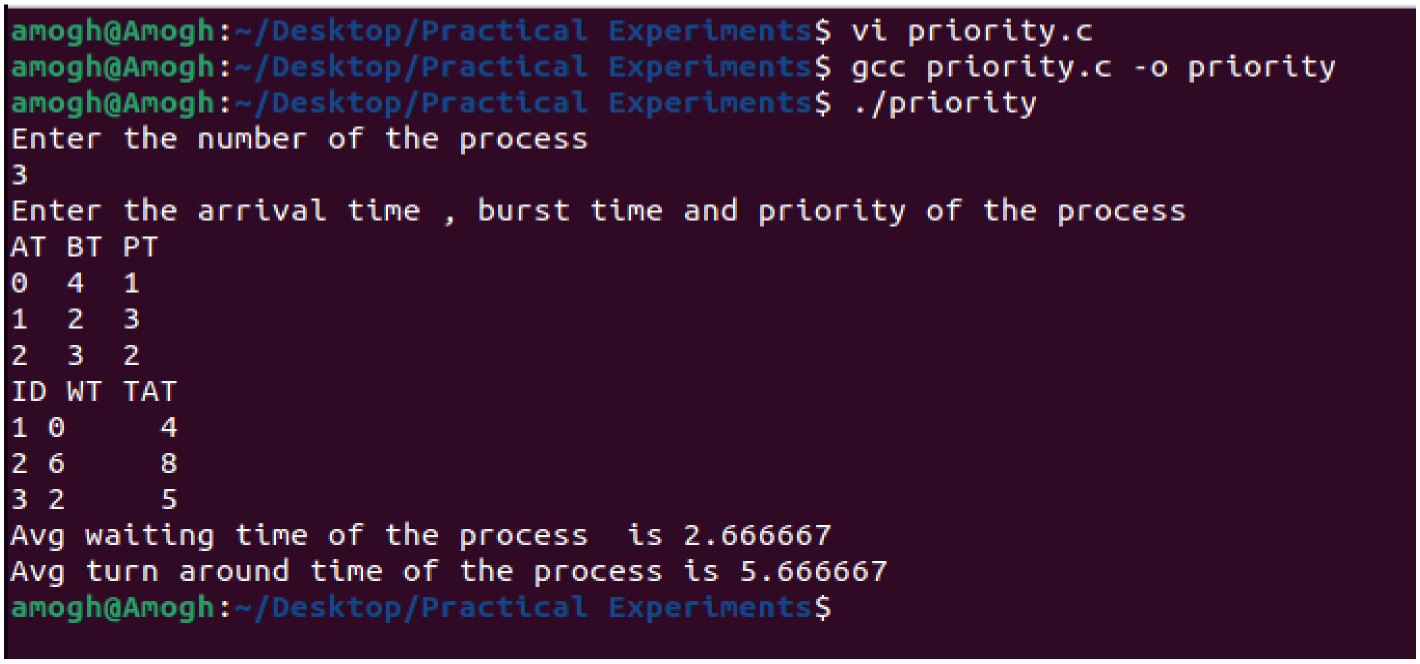


```c
#include<stdio.h>
struct process
{
    int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
    int n,temp[10],t,count=0,short_p;
    float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of the process\n");
    scanf("%d",&n);
    printf("Enter the arrival time , burst time and priority of the process\n");
    printf("AT BT PT\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

        // copying the burst time in
        // a temp array fot futher use
        temp[i]=a[i].BT;
    }

    // we initialize the burst time
    // of a process with maximum
    a[9].PT=10000;

    for(t=0;count!=n;t++)
    {
        short_p=9;
        for(int i=0;i<n;i++)
        {
            if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
            {
                short_p=i;
            }
        }

        a[short_p].BT=a[short_p].BT-1;

        // if any process is completed
        if(a[short_p].BT==0)
        {
```



```c
    for(t=0;count!=n;t++)
    {
        short_p=9;
        for(int i=0;i<n;i++)
        {
            if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
            {
                short_p=i;
            }
        }

        a[short_p].BT=a[short_p].BT-1;

        // if any process is completed
        if(a[short_p].BT==0)
        {
```

```c
            // one process is completed
            // so count increases by 1
            count++;
            a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
            a[short_p].TAT=t+1-a[short_p].AT;

            // total calculation
            total_WT=total_WT+a[short_p].WT;
            total_TAT=total_TAT+a[short_p].TAT;

        }
    }

    Avg_WT=total_WT/n;
    Avg_TAT=total_TAT/n;

    // printing of the answer
    printf("ID WT TAT\n");
    for(int i=0;i<n;i++)
    {
        printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);
    }

    printf("Avg waiting time of the process  is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);

    return 0;
}
```



```
amogh@Amogh:~/Desktop/Practical Experiments$ vi priority.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc priority.c -o priority
amogh@Amogh:~/Desktop/Practical Experiments$ ./priority
Enter the number of the process
3
Enter the arrival time , burst time and priority of the process
AT BT PT
0  4  1
1  2  3
2  3  2
ID WT TAT
1 0     4
2 6     8
3 2     5
Avg waiting time of the process  is 2.666667
Avg turn around time of the process is 5.666667
amogh@Amogh:~/Desktop/Practical Experiments$
```