NAME: Spandan Mukherjee
Registration Number: 21BCE1132
Subject: Operating Systems
Slot: F2
Experiment: LAB 6 CPU Scheduling

1.

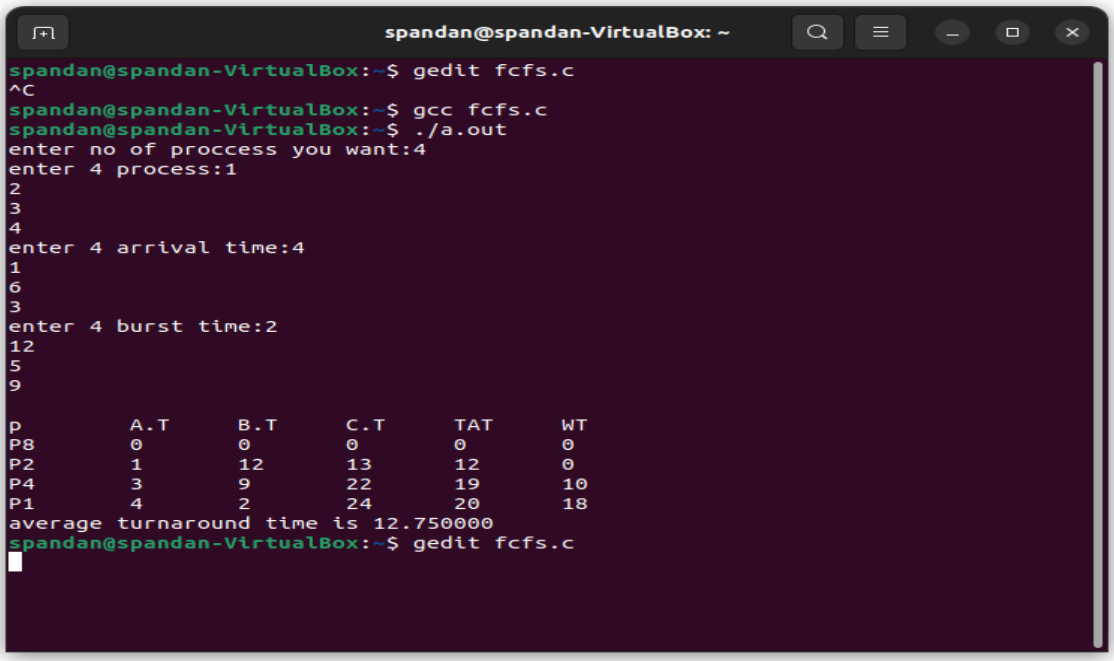
(i) Implement FCFS scheduling with different arrival time and calculate waiting time, CT,TAT for all the processes.

FCFS WITH ARRIVAL TIME (Non-Preemptive)

```
#include<stdio.h>
int main()
{
   int  p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;
   float awt=0,atat=0;
   printf("enter no of proccess you want:");
   scanf("%d",&n);
   printf("enter %d process:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&p[i]);
   }
   printf("enter %d arrival time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&at[i]);
   }
   printf("enter %d burst time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&bt[i]);
   }
   // sorting at,bt, and process according to at
   for(i=0;i<n;i++)
   {
    for(j=0;j<(n-i);j++)
   {
     if(at[j]>at[j+1])
    {
       temp=p[j+1];
       p[j+1]=p[j];
       p[j]=temp;
       temp=at[j+1];
       at[j+1]=at[j];
       at[j]=temp;
       temp=bt[j+1];
       bt[j+1]=bt[j];
       bt[j]=temp;
     }
```

```
    }
   }
   /* calculating 1st ct */
   ct[0]=at[0]+bt[0];
   /* calculating 2 to n ct */
   for(i=1;i<n;i++)
   {
     //when proess is ideal in between i and i+1
     temp=0;
    if(ct[i-1]<at[i])
    {
       temp=at[i]-ct[i-1];
    }
    ct[i]=ct[i-1]+bt[i]+temp;
   }
   /* calculating tat and wt */
   printf("\np\t A.T\t B.T\t C.T\t TAT\t WT");
   for(i=0;i<n;i++)
   {
   tat[i]=ct[i]-at[i];
   wt[i]=tat[i]-bt[i];
   atat+=tat[i];
   awt+=wt[i];
   }
   atat=atat/n;
   awt=awt/n;
   for(i=0;i<n;i++)
   {
     printf("\nP%d\t %d\t %d\t %d \t %d \t %d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
   }
   printf("\naverage turnaround time is %f",atat);

   printf("\naverage wating time is %f",awt);
   return 0;
}
```

(ii)

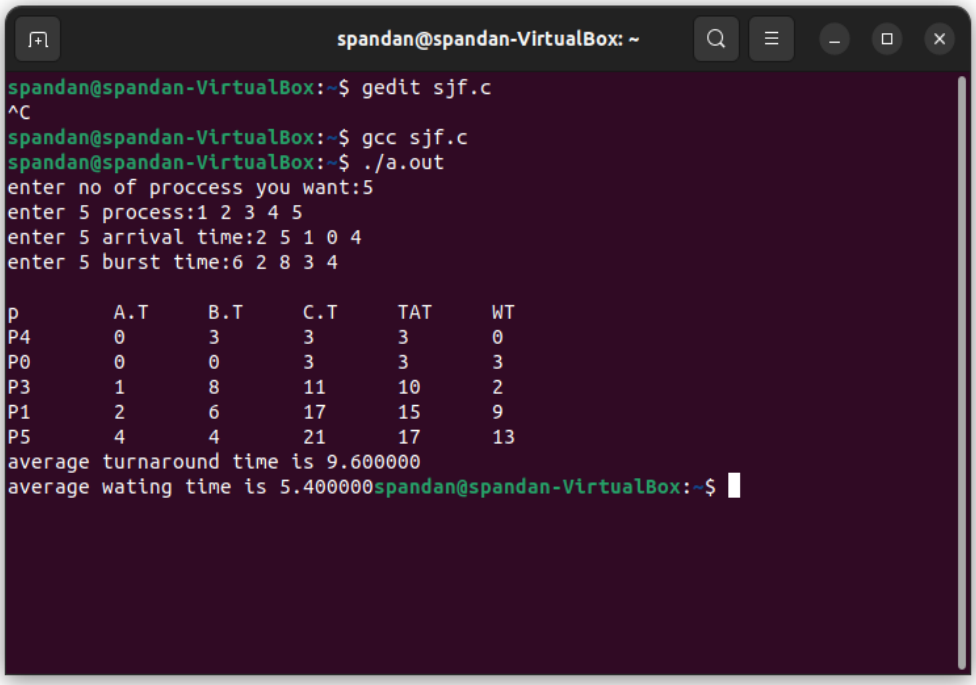


Implement SJF scheduling with different arrival time and calculate waiting time, CT,TAT for all the processes .

SJF Non-Preemptive with Arrival Time

```
#include<stdio.h>
int main()
{
   int  p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;
   float awt=0,atat=0;
   printf("enter no of proccess you want:");
   scanf("%d",&n);
   printf("enter %d process:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&p[i]);
   }
   printf("enter %d arrival time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&at[i]);
   }
   printf("enter %d burst time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&bt[i]);
   }
   // sorting at,bt, and process according to at
   for(i=0;i<n;i++)
   {
    for(j=0;j<(n-i);j++)
```

```
    {
      if(at[j]>at[j+1])
     {
        temp=p[j+1];
        p[j+1]=p[j];
        p[j]=temp;
        temp=at[j+1];
        at[j+1]=at[j];
        at[j]=temp;
        temp=bt[j+1];
        bt[j+1]=bt[j];
        bt[j]=temp;
      }
     }
    }
    /* calculating 1st ct */
    ct[0]=at[0]+bt[0];
    /* calculating 2 to n ct */
    for(i=1;i<n;i++)
    {
      //when proess is ideal in between i and i+1
      temp=0;
     if(ct[i-1]<at[i])
     {
        temp=at[i]-ct[i-1];
     }
     ct[i]=ct[i-1]+bt[i]+temp;
    }
    /* calculating tat and wt */
    printf("\np\t A.T\t B.T\t C.T\t TAT\t WT");
    for(i=0;i<n;i++)
    {
    tat[i]=ct[i]-at[i];
    wt[i]=tat[i]-bt[i];
    atat+=tat[i];
    awt+=wt[i];
    }
    atat=atat/n;
    awt=awt/n;
    for(i=0;i<n;i++)
    {
      printf("\nP%d\t %d\t %d\t %d \t %d \t %d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
    }
    printf("\naverage turnaround time is %f",atat);

    printf("\naverage wating time is %f",awt);
    return 0;
}
```



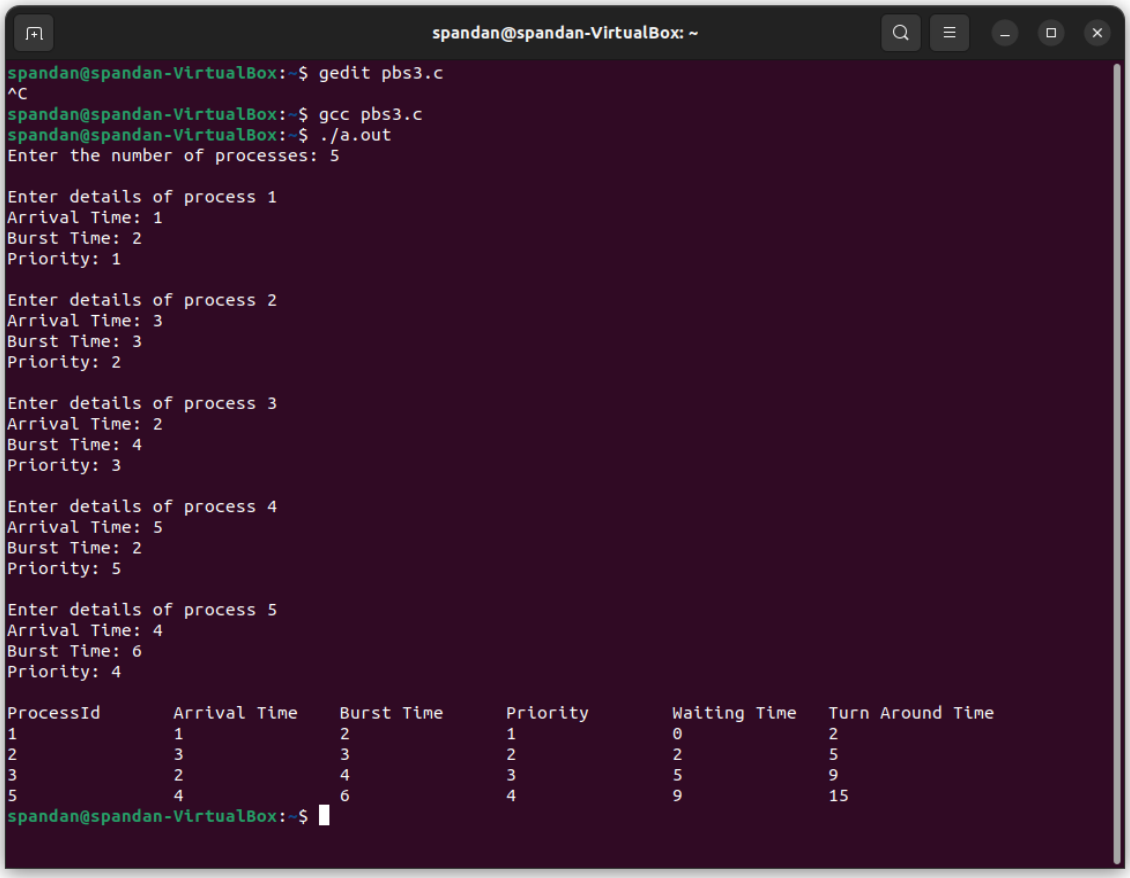
(iii) Implement PBS scheduling with different arrival time and calculate waiting time, CT,TAT for all the processes.

CODE:

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

struct process{
    int processId;
    int arrivalTime;
    int burstTime;
    int priority;
    int waitingTime;
    int turnAroundTime;
};

int main(){
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct process p[n];
    for(int i = 0; i < n; i++){
        printf("\nEnter details of process %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &p[i].arrivalTime);
        printf("Burst Time: ");
        scanf("%d", &p[i].burstTime);
```

```
        printf("Priority: ");
        scanf("%d", &p[i].priority);
        p[i].processId = i + 1;
    }

    // Sort processes by priority (if two processes have same priority, then consider their arrival time)
    for(int i = 0; i < n - 1; i++){
        for(int j = i + 1; j < n; j++){
            if(p[i].priority > p[j].priority){
                struct process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
            else if(p[i].priority == p[j].priority){
                if(p[i].arrivalTime > p[j].arrivalTime){
                    struct process temp = p[i];
                    p[i] = p[j];
                    p[j] = temp;
                }
            }
        }
    }

    // Calculate waiting time and turn around time of each process
    p[0].waitingTime = 0;
    p[0].turnAroundTime = p[0].burstTime;
    for(int i = 1; i < n; i++){
        p[i].waitingTime = p[i - 1].waitingTime + p[i - 1].burstTime;
        p[i].turnAroundTime = p[i].waitingTime + p[i].burstTime;
    }

    printf("\nProcessId\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurn Around Time");
    for(int i = 0; i < n; i++){
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", p[i].processId, p[i].arrivalTime, p[i].burstTime,
p[i].priority, p[i].waitingTime, p[i].turnAroundTime);
    }
```

```
    return 0;
}
```

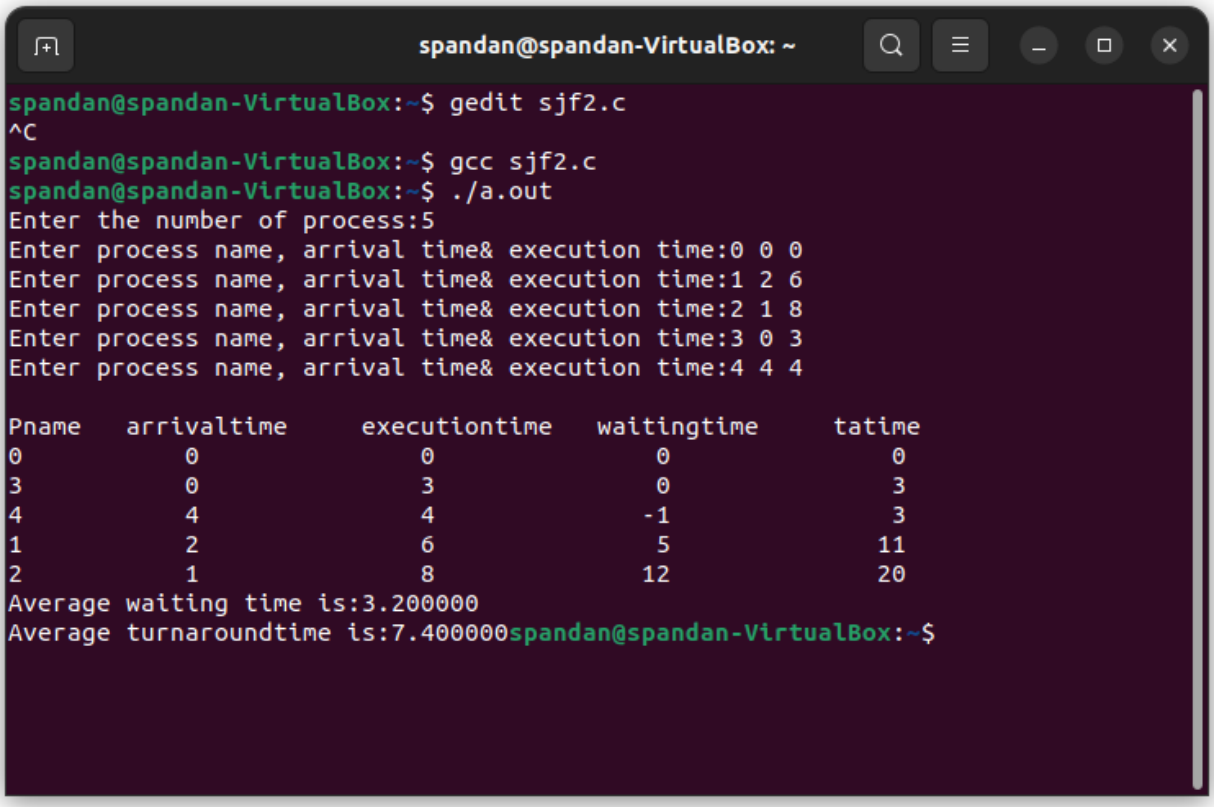


2. Implement the pre-emptive version of SJF and PBS. calculate waiting time, response time, CT,TAT for all the processes.

SJF Preemptive with Arrival time

```
#include<stdio.h>

#include<string.h>
void main()
{
   int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
   int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   //clrscr();
```

```
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
       printf("Enter process name, arrival time& execution time:");
       //flushall();
       scanf("%s%d%d",pn[i],&at[i],&et[i]);
    }
    for(i=0; i<n; i++)
       for(j=0; j<n; j++)
       {
          if(et[i]<et[j])
          {
             temp=at[i];
             at[i]=at[j];
             at[j]=temp;
             temp=et[i];
             et[i]=et[j];
             et[j]=temp;
             strcpy(t,pn[i]);
             strcpy(pn[i],pn[j]);
             strcpy(pn[j],t);
          }
       }
    for(i=0; i<n; i++)
    {
       if(i==0)
          st[i]=at[i];
       else
          st[i]=ft[i-1];
       wt[i]=st[i]-at[i];
       ft[i]=st[i]+et[i];
       ta[i]=ft[i]-at[i];
       totwt+=wt[i];
       totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
    printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
    for(i=0; i<n; i++)
       printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);

}
```

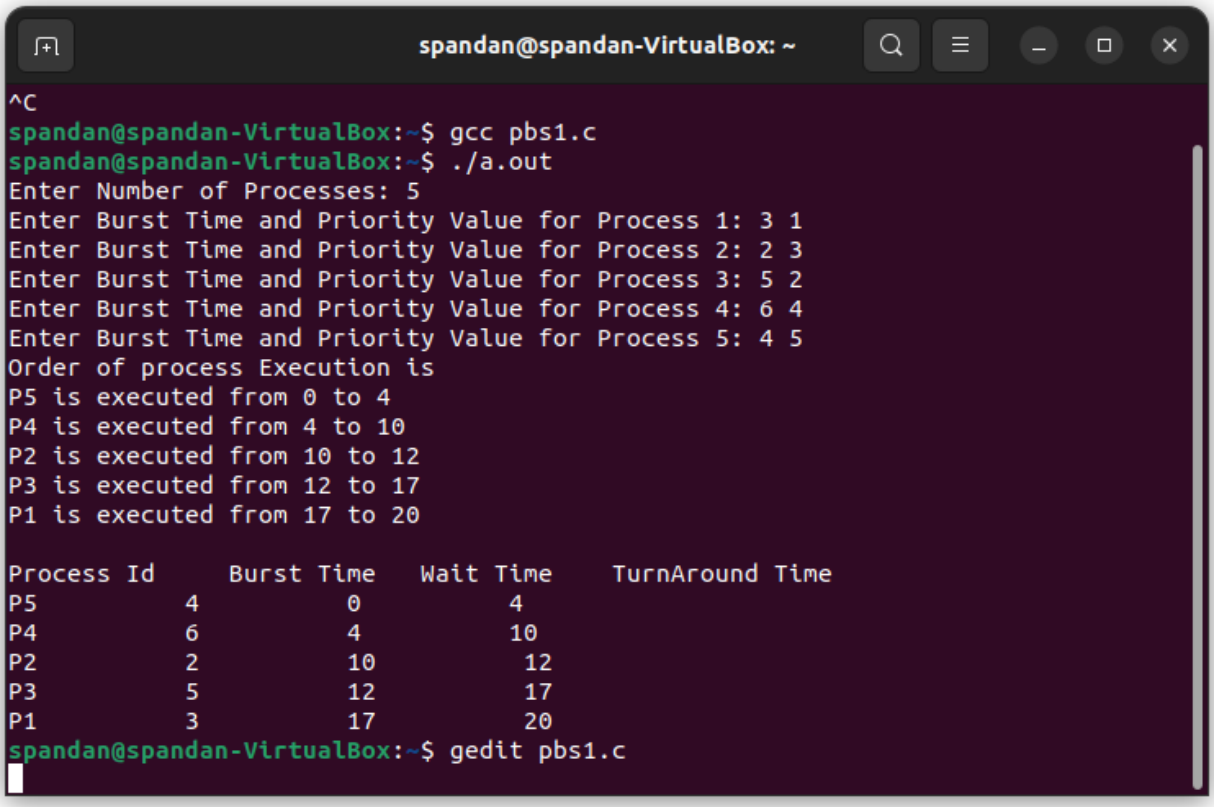


Priority Based Scheduling

```
#include <stdio.h>


//Function to swap two variables
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n;
```

```
printf("Enter Number of Processes: ");
scanf("%d",&n);

// b is array for burst time, p for priority and index for process id
int b[n],p[n],index[n];
for(int i=0;i<n;i++)
{
    printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
    scanf("%d %d",&b[i],&p[i]);
    index[i]=i+1;
}
for(int i=0;i<n;i++)
{
    int a=p[i],m=i;

    //Finding out highest priority element and placing it at its desired position
    for(int j=i;j<n;j++)
    {
        if(p[j] > a)
        {
            a=p[j];
            m=j;
        }
    }

    //Swapping processes
```

```
            swap(&p[i], &p[m]);
            swap(&b[i], &b[m]);
            swap(&index[i],&index[m]);
        }

        // T stores the starting time of process
        int t=0;

        //Printing scheduled process
        printf("Order of process Execution is\n");
        for(int i=0;i<n;i++)
        {
            printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
            t+=b[i];
        }
        printf("\n");
        printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
        int wait_time=0;
        for(int i=0;i<n;i++)
        {
            printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);
            wait_time += b[i];
        }
        return 0;
    }
```



3. Implement the RR scheduling. calculate waiting time, response time, CT,TAT for all the processes.

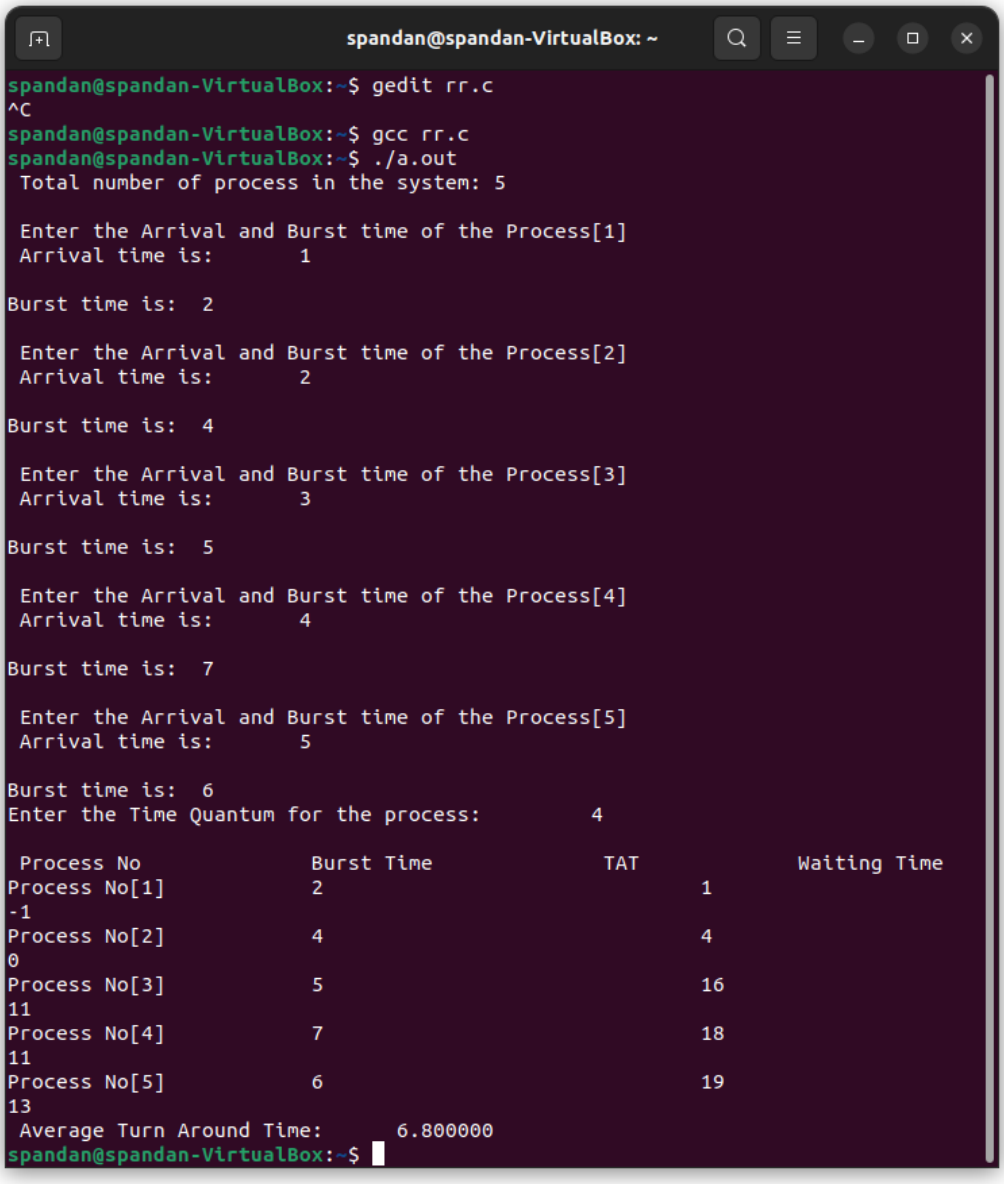
CODE:

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

```
void main()
{
   // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
   {
      y--; //decrement the process no.
      printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
      wt = wt+sum-at[i]-bt[i];
      tat = tat+sum-at[i];
      count =0;
   }
   if(i==NOP-1)
   {
      i=0;
   }
   else if(at[i+1]<=sum)
   {
```

```
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);

return 0;

}
```

4. Create 3 threads and ensure that each thread does (FCFS, non pre-emptive SJF, non pre-emptive PBS) and



returns the process id which has the longest waiting time. Display the process id with longest waiting time in main().

Code:

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

#define MAX_PROCESSES 100

typedef struct {
  int arrival_time;
  int burst_time;
  int priority;
} Process;

void FCFS(Process processes[], int num_processes) {
  int waiting_times[MAX_PROCESSES];
  int total_waiting_time = 0;

  waiting_times[0] = 0;
  for (int i = 1; i < num_processes; i++) {
    waiting_times[i] = processes[i-1].burst_time + waiting_times[i-1];
  }

  printf("Process\t\tWaiting Time\n");
  for (int i = 0; i < num_processes; i++) {
    total_waiting_time += waiting_times[i];
    printf("%d\t\t%d\n", i+1, waiting_times[i]);
  }

  printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / num_processes);
}


int compare_processes(const void *a, const void *b) {
  Process *p1 = (Process *)a;
  Process *p2 = (Process *)b;
  return p1->burst_time - p2->burst_time;
}

void SJF(Process processes[], int num_processes) {
  int waiting_times[MAX_PROCESSES];
  int total_waiting_time = 0;
  bool processed[MAX_PROCESSES];

  qsort(processes, num_processes, sizeof(Process), compare_processes);

  for (int i = 0; i < num_processes; i++) {
    processed[i] = false;
```

```
  }

  waiting_times[0] = 0;
  for (int i = 1; i < num_processes; i++) {
    int j = i - 1;
    while (j >= 0 && !processed[j]) {
      waiting_times[i] += processes[j].burst_time;
      j--;
    }
  }

  printf("Process\t\tWaiting Time\n");
  for (int i = 0; i < num_processes; i++) {
    total_waiting_time += waiting_times[i];
    processed[i] = true;
    printf("%d\t\t%d\n", i+1, waiting_times[i]);
  }

  printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / num_processes);
}



void PriorityScheduling(Process processes[], int num_processes) {
  int waiting_times[MAX_PROCESSES];
  int total_waiting_time = 0;
  bool processed[MAX_PROCESSES];

  qsort(processes, num_processes, sizeof(Process), compare_processes);

  for (int i = 0; i < num_processes; i++) {
    processed[i] = false;
  }

  waiting_times[0] = 0;
  for (int i = 1; i < num_processes; i++) {
    int j = i - 1;
    while (j >= 0 && !processed[j]) {
      waiting_times[i] += processes[j].burst_time;
      j--;
    }
  }

  printf("Process\t\tWaiting Time\n");
  for (int i = 0; i < num_processes; i++) {
    total_waiting_time += waiting_times[i];
    processed[i] = true;
    printf("%d\t\t%d\n", i+1, waiting_times[i]);
  }

  printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / num_processes);
}
```

```
int main()
{



    pthread_t thread_id1, thread_id2, thread_id3;
    printf("Before Thread\n");
    pthread_create(&thread_id1, NULL, FCFS, NULL);
    pthread_join(thread_id1, NULL);
    printf("After Thread\n");


    printf("Before Thread\n");
    pthread_create(&thread_id2, NULL, SJF, NULL);
    pthread_join(thread_id2, NULL);
    printf("After Thread\n");


    printf("Before Thread\n");
    pthread_create(&thread_id3, NULL, PriorityScheduling, NULL);
    pthread_join(thread_id3, NULL);
    printf("After Thread\n");




    return 0;
}
```

CODE 2:

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

#define NUM_THREADS 3

int waiting_time[NUM_THREADS];

void *fcfs(void *arg) {
    int process_id = *((int *) arg);
    waiting_time[process_id] = process_id * 2;
    sleep(waiting_time[process_id]);
    return (void *) &waiting_time[process_id];
}

void *sjf(void *arg) {
    int process_id = *((int *) arg);
    waiting_time[process_id] = (NUM_THREADS - process_id) * 2;
    sleep(waiting_time[process_id]);
    return (void *) &waiting_time[process_id];
}

void *pbs(void *arg) {
    int process_id = *((int *) arg);
    waiting_time[process_id] = (process_id % 2 == 0) ? 4 : 6;
    sleep(waiting_time[process_id]);
    return (void *) &waiting_time[process_id];
}

int main() {
    pthread_t threads[NUM_THREADS];
    int process_id[NUM_THREADS];
    int *result;
    int max_waiting_time = 0;
    int max_process_id = 0;
    for (int i = 0; i < NUM_THREADS; i++) {
        process_id[i] = i;
        if (i == 0) {
            pthread_create(&threads[i], NULL, fcfs, (void *) &process_id[i]);
        } else if (i == 1) {
            pthread_create(&threads[i], NULL, sjf, (void *) &process_id[i]);
        } else {
            pthread_create(&threads[i], NULL, pbs, (void *) &process_id[i]);
        }
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], (void **) &result);
        if (*result > max_waiting_time) {
            max_waiting_time = *result;
            max_process_id = process_id[i];
        }
```

```
    }
    printf("Process with longest waiting time: %d\n", max_process_id);
    return 0;
}
```