SchedCraft.

Simulating Process Scheduling Algorithms in C

Operating System (IT5403)

Mini Project

Done by

Rameez Akther M - 2022510025

Arun Kumar S - 2022510039

B.Tech AI&DS

IT Department

MIT Campus

Anna University

Abstract

This project presents an implementation of various process scheduling algorithms in the C programming language. The implemented algorithms include First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Round Robin (RR), Priority Preemptive, and Priority Non-Preemptive scheduling. Each algorithm is simulated with user-provided inputs for the number of processes, arrival times, burst times, and in some cases, priority levels. The program calculates and displays key metrics for each process, including completion time, turnaround time, waiting time, and response time. It also calculates and displays the average turnaround time, waiting time, and response time for each scheduling algorithm. Additionally, the program generates a Gantt chart to visually represent the order and duration of process execution. This project serves as a practical tool for understanding and comparing the performance of different process scheduling algorithms in operating systems.

Objective

The objective of the process scheduling algorithms implemented in your project is to manage the execution of processes in an operating system. Each algorithm has a specific goal:

1. **First-Come, First-Served (FCFS)**: This algorithm aims to execute processes in the order they arrive. It's simple and easy to understand, but it can lead to poor performance if long processes arrive before short ones.
2. **Shortest Job First (SJF)**: The objective of this algorithm is to minimize waiting time by executing the shortest jobs first. However, it requires knowing the burst time of each process in advance.
3. **Shortest Remaining Time First (SRTF)**: This is a preemptive version of SJF. Its goal is to always execute the process with the shortest remaining burst time. This can lead to lower waiting times than SJF, but it requires more context switches.
4. **Round Robin (RR)**: The objective of this algorithm is to ensure fair process scheduling by giving each process a fixed time slice (or "quantum") in which to execute. If a process doesn't finish within its time slice, it's moved to the back of the queue.
5. **Priority Scheduling (Preemptive and Non-Preemptive)**: These algorithms aim to execute processes based on their priority. In the preemptive version, a higher-priority process can interrupt a lower-priority one. In the non-preemptive version, a process runs to completion once it starts.

By implementing these algorithms, the project provides a practical tool for understanding how different scheduling strategies can impact process execution, turnaround time, waiting time, and response time in an operating system. It also allows for visualizing the order and duration of process execution through Gantt charts. This can be useful for both educational purposes and for designing efficient operating systems.

Scope

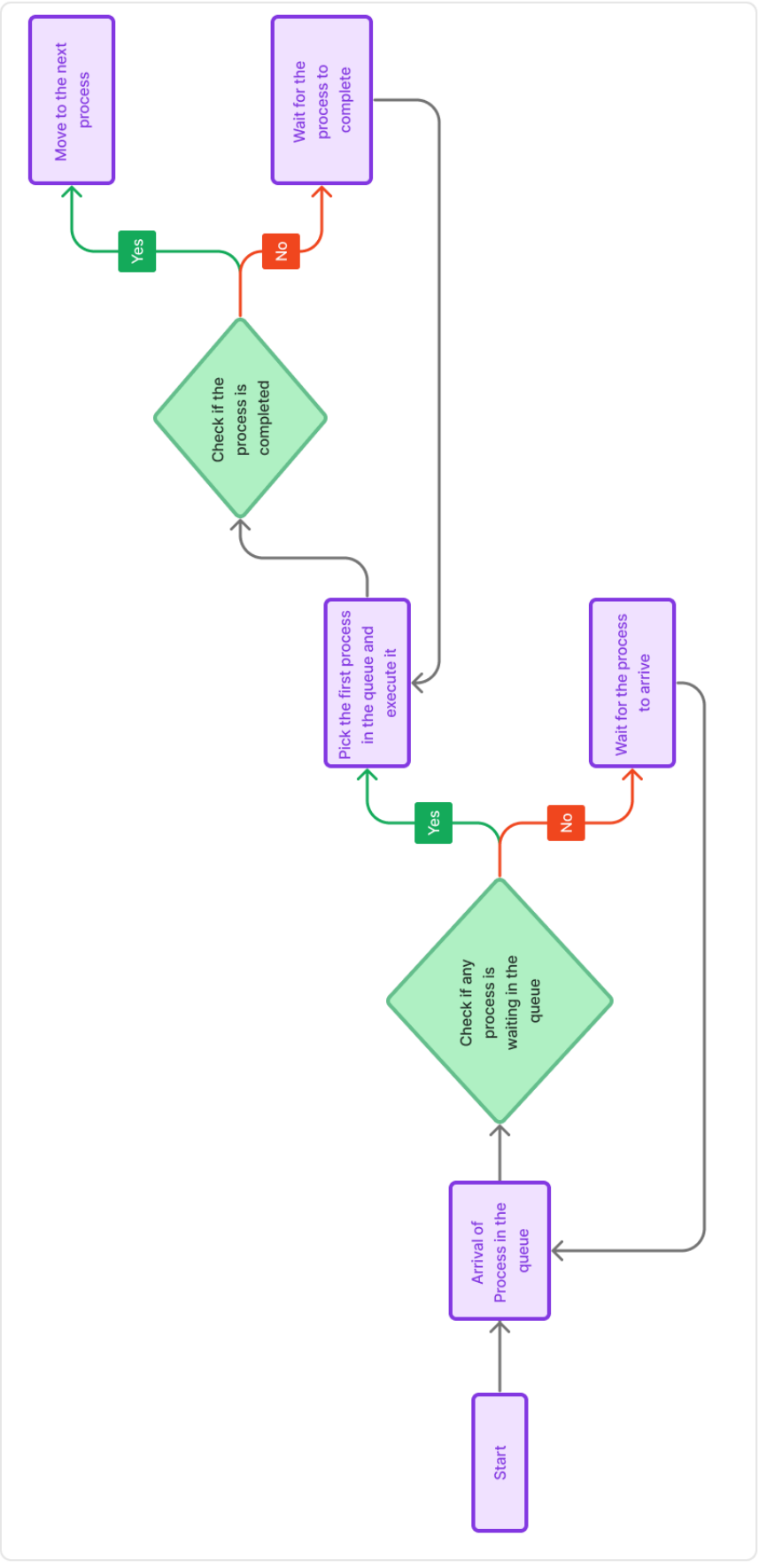
The scope of this project is quite broad and impactful, as it covers several key areas:

1. **Educational Tool**: This project serves as a valuable educational tool for students and professionals learning about operating systems and process scheduling algorithms. It provides practical, hands-on experience with various scheduling algorithms, helping users understand their workings and trade-offs.
2. **Performance Analysis**: By implementing and simulating different scheduling algorithms, this project allows for a comparative analysis of their performance in terms of turnaround time, waiting time, and response time. This can be useful in both academic research and real-world applications.
3. **Algorithm Visualization**: The project includes the generation of Gantt charts, which are a powerful tool for visualizing the scheduling and execution of processes. This can aid in understanding the behavior of different scheduling algorithms.
4. **Basis for Further Development**: This project can be extended or adapted to simulate more complex scheduling scenarios, incorporate additional scheduling algorithms, or even be integrated into a larger operating system simulation or development project.
5. **Real-world Applications**: Understanding these algorithms and their implications can be beneficial in real-world applications such as improving system efficiency, process management, load balancing, and more in various computing environments.

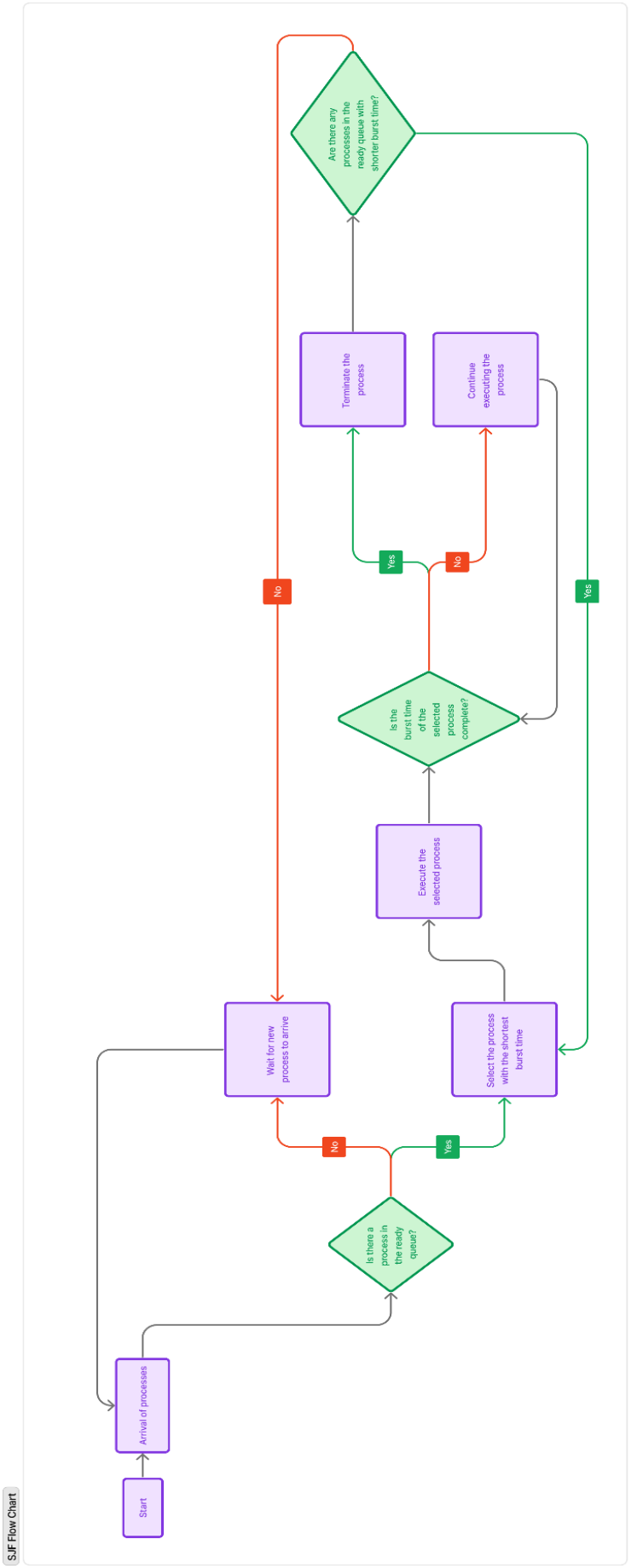
In summary, the scope of this project extends from educational use to performance analysis, algorithm visualization, and real-world applications, making it a versatile tool in the field of computer science, particularly in the study and application of operating systems.

Diagram

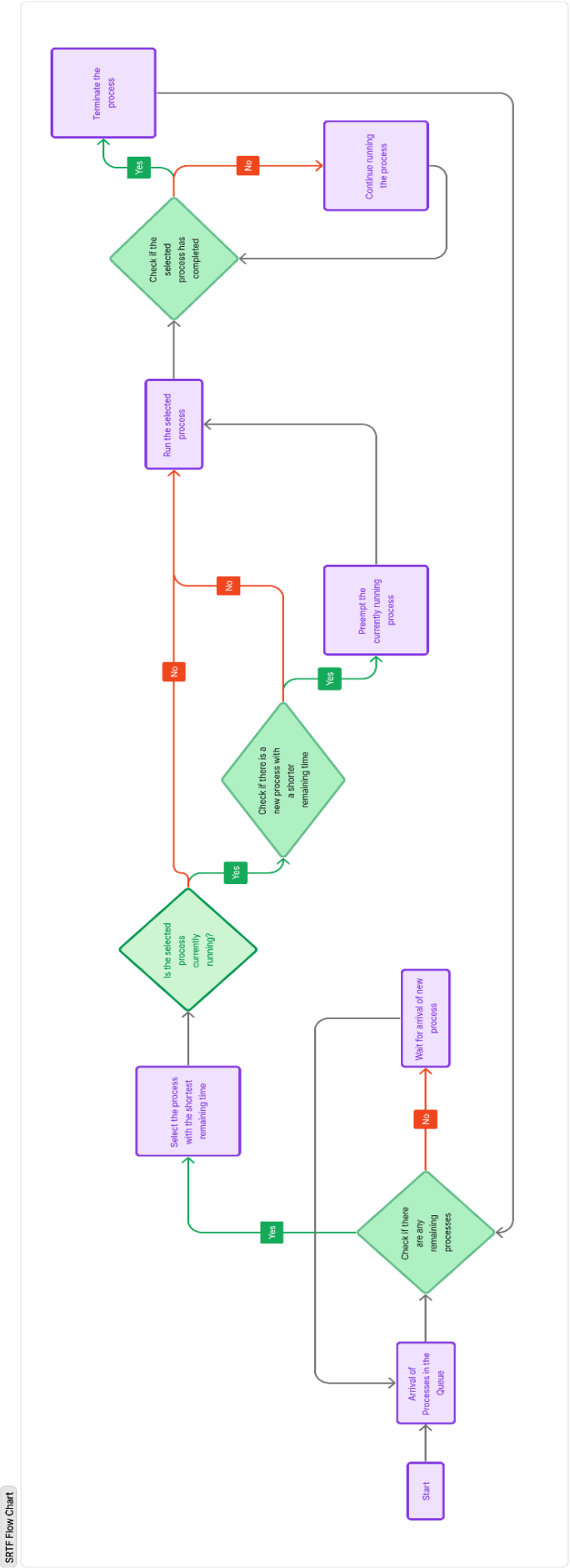
1. FCFS (First Come First Serve) FlowChart



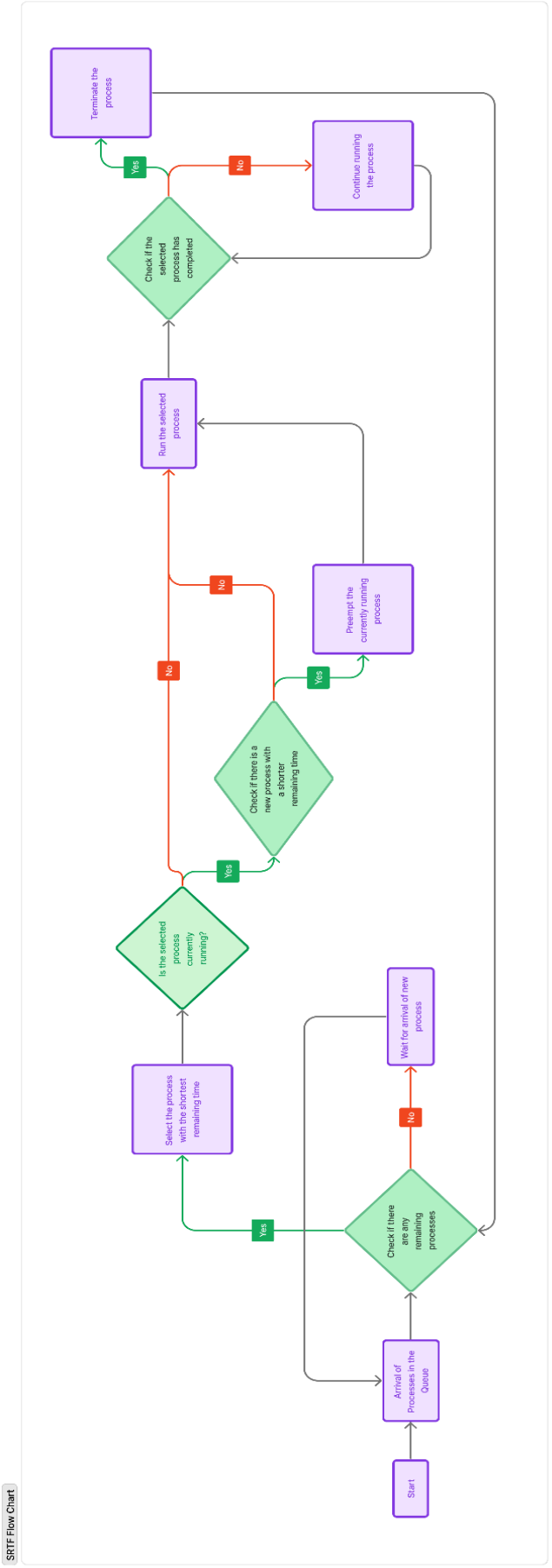
2. SJF (Shortest Job First) FlowChart



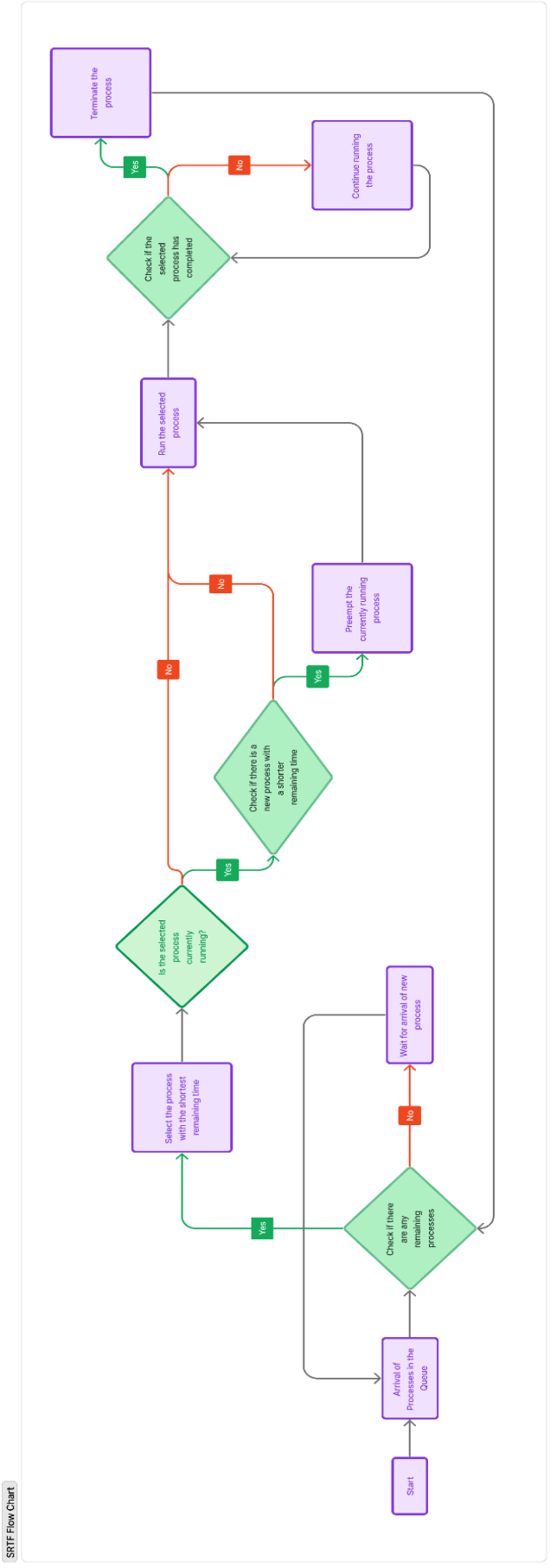
3. SRTF (Shortest Remaining Time First) FlowChart



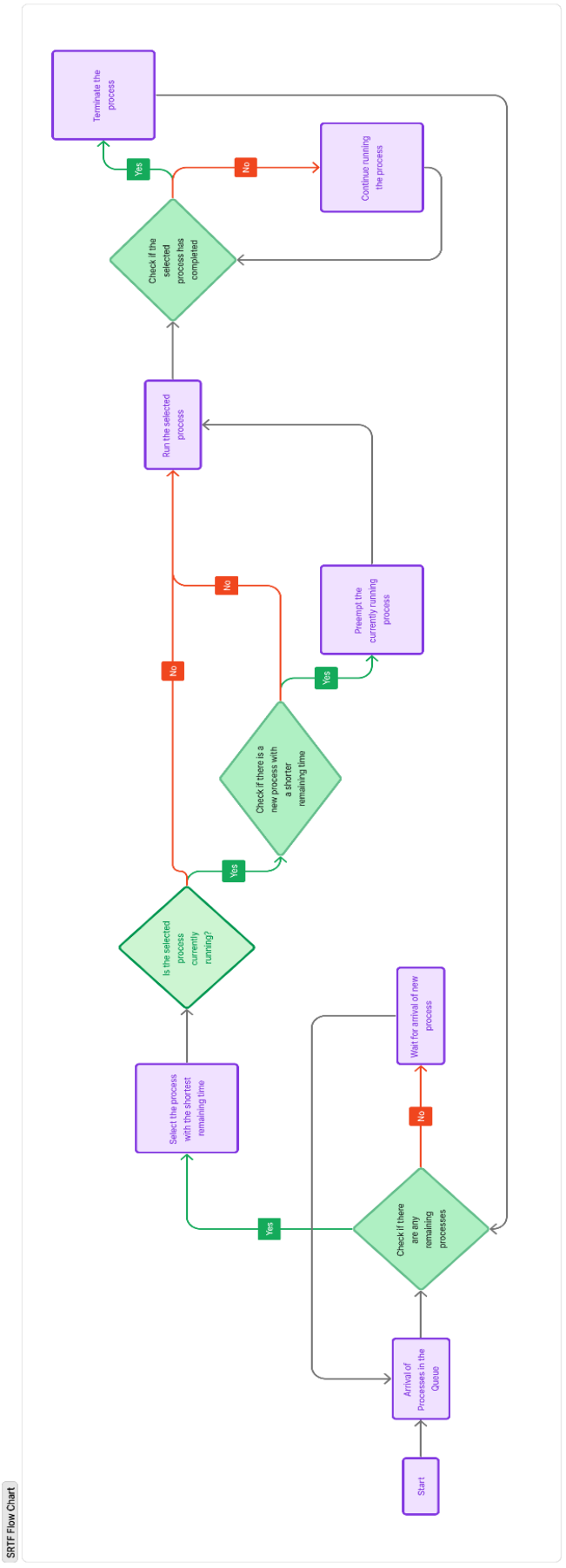
4. Priority Preemptive FlowChart



5. Priority Non Preemptive FlowChart



6. Round Robin FlowChart



Algorithm

First-Come, First-Served (FCFS)

1. Input the number of processes (pn), arrival times (at), and burst times (bt) for each process.
2. Sort the processes based on their arrival times in ascending order.
3. Initialize variables for completion time (com), turnaround time (tat), waiting time (wt), average turnaround time (avgtat), average waiting time (avgwt), and CPU idle time (idle_time).
4. Calculate completion times for each process:
    a. For the first process, completion time = arrival time + burst time.
    b. For subsequent processes, if the previous process's completion time is less than the current process's arrival time, set idle time = current process's arrival time - previous process's completion time, and then set completion time = current process's arrival time + burst time.
    c. Otherwise, set completion time = previous process's completion time + burst time.
5. Calculate turnaround time for each process: turnaround time = completion time - arrival time.
6. Calculate waiting time for each process: waiting time = turnaround time - burst time.
7. Calculate average turnaround time: sum of turnaround times / number of processes.
8. Calculate average waiting time: sum of waiting times / number of processes.
9. Print the process table showing process number, arrival time, burst time, completion time, turnaround time, and waiting time for each process.
10. Print the average turnaround time, average waiting time, and CPU idle time.
11. Print the Gantt chart showing the timeline of process execution.

Shortest Job First (SJF)

1. Input the number of processes (n), arrival times (Arr), and burst times (BT) for each process.
2. Create arrays for copy of burst times (Bt), completion times (Ct), process numbers (P), turnaround times (TAT), and waiting times (WT).
3. Initialize variables for current time and Gantt chart.
4. While there are still processes remaining to execute:
    a. Find the process with the shortest burst time that has arrived and not yet executed.
    b. Update current time and completion time for the selected process.
    c. Update Gantt chart with the selected process's execution.
5. Calculate turnaround time for each process: turnaround time = completion time - arrival time.

6. Calculate waiting time for each process: waiting time = turnaround time - burst time.
7. Print the process table showing process ID, arrival time, burst time, completion time, turnaround time, and waiting time for each process.
8. Print the average turnaround time and average waiting time.
9. Print the Gantt chart showing the timeline of process execution.

Shortest Remaining Time First (SRTF)

1. Input the number of processes (n), arrival times (Arr), and burst times (BT) for each process.
2. Create arrays for copy of burst times (Bt), completion times (Ct), response times (Rt), process numbers (P), turnaround times (TAT), and waiting times (WT).
3. Initialize variables for current time, count of completed processes, and previous process index.
4. While there are still processes remaining to execute:
    a. Find the process with the shortest remaining burst time that has arrived and not yet executed.
    b. Update remaining burst time for the selected process.
    c. Update response time if it's the first time the process is being executed.
    d. Update Gantt chart with the selected process's execution.
    e. Update completion time and increment count of completed processes if the process has finished.
5. Calculate turnaround time for each process: turnaround time = completion time - arrival time.
6. Calculate waiting time for each process: waiting time = turnaround time - burst time.
7. Print the process table showing process ID, arrival time, burst time, completion time, turnaround time, waiting time, and response time for each process.
8. Print the average turnaround time, average waiting time, and average response time.
9. Print the Gantt chart showing the timeline of process execution.

Round Robin (RR)

1. Input the number of processes (n), arrival times (Arr), burst times (BT), and time quantum (tq) for each process.
2. Create arrays for copy of burst times (Bt), completion times (Ct), response times (Rt), ready queue (ready), process numbers (P), turnaround times (TAT), and waiting times (WT).
3. Initialize variables for current time, ready queue front and rear, and a flag for indicating idle time.
4. While there are still processes remaining to execute:
    a. Add processes arriving at the current time to the ready queue.

   b. If the ready queue is empty, add an idle entry to the Gantt chart.
   c. Process the ready queue using round robin:
      i. If the remaining burst time is less than or equal to the time quantum, process the entire burst time.
      ii. If the remaining burst time is greater than the time quantum, process the time quantum and add the process back to the ready queue.
   d. Update completion time, response time, and check for process completion.
5. Calculate turnaround time for each process: turnaround time = completion time - arrival time.
6. Calculate waiting time for each process: waiting time = turnaround time - burst time.
7. Print the process table showing process ID, arrival time, burst time, completion time, turnaround time, waiting time, and response time for each process.
8. Print the average turnaround time, average waiting time, and average response time.
9. Print the Gantt chart showing the timeline of process execution.

Priority Preemptive Scheduling

1. Input the number of processes (n), arrival times (Arr), burst times (BT), and priorities (PR) for each process.
2. Create arrays for copy of burst times (Bt), completion times (Ct), response times (Rt), priorities (Pr), process numbers (P), turnaround times (TAT), and waiting times (WT).
3. Initialize variables for current time, previous process index, and a flag for indicating idle time.
4. Input the scheduling option for priority comparison: Lowest number Highest priority or Lowest number Lowest priority.
5. While there are still processes remaining to execute:
   a. Find the process with the highest or lowest priority (based on the scheduling option) that has arrived and not yet executed.
   b. Update remaining burst time for the selected process.
   c. Update Gantt chart with the selected process's execution.
   d. Update response time if it's the first time the process is being executed.
   e. Update completion time and check for process completion.
6. Calculate turnaround time for each process: turnaround time = completion time - arrival time.
7. Calculate waiting time for each process: waiting time = turnaround time - burst time.
8. Print the process table showing process ID, priority, arrival time, burst time, completion time, turnaround time, and waiting time for each process.
9. Print the average turnaround time, average waiting time, and average response time.

10. Print the Gantt chart showing the timeline of process execution.

Priority Non-Preemptive Scheduling

1. Input the number of processes (n), arrival times (Arr), burst times (BT), and priorities (PR) for each process.
2. Create arrays for copy of priorities (Pr), completion times (Ct), process numbers (P), turnaround times (TAT), and waiting times (WT).
3. Initialize variables for current time, a flag for indicating idle time, and a choice for priority comparison.
4. Input the scheduling option for priority comparison: Lowest number Highest priority or Lowest number Lowest priority.
5. While there are still processes remaining to execute:
    a. Find the process with the highest or lowest priority (based on the scheduling option) that has arrived and not yet executed.
    b. Update Gantt chart with the selected process's execution.
    c. Update completion time and check for process completion.
6. Calculate turnaround time for each process: turnaround time = completion time - arrival time.
7. Calculate waiting time for each process: waiting time = turnaround time - burst time.
8. Print the process table showing process ID, priority, arrival time, burst time, completion time, turnaround time, and waiting time for each process.
9. Print the average turnaround time and average waiting time.
10. Print the Gantt chart showing the timeline of process execution.

Source Code

File name: FCFS.c
(The following code simulates First Come First Serve process scheduling algorithm)

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

int main() {
    printf("\n---------------------------------------\n\t     FCFS
Scheduling\n---------------------------------------\n");
    int pn, at[20], bt[20], i, tat[20], com[20], wt[20], temp, j;
    float avg = 0, avg2 = 0, avgtat, avgwt;
    printf("\nEnter the number of processes for FCFS SCHEDULING: ");
    scanf("%d", &pn);
    printf("\nEnter the arrival times: ");
    for (i = 0; i < pn; i++)
        scanf("%d", &at[i]);
    printf("\nEnter the burst times: ");
    for (i = 0; i < pn; i++)
        scanf("%d", &bt[i]);
    printf("\n\n");
    for (i = 0; i < pn; i++) {
        for (j = i + 1; j < pn; j++) {
            if (at[i] > at[j]) {
                temp = at[i];
                at[i] = at[j];
                at[j] = temp;

                temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;
            }
        }
    }

    int idle_time = 0;
    com[0] = bt[0] + at[0];
    tat[0] = bt[0];
    wt[0] = 0;
    for (i = 1; i < pn; i++) {
        if (com[i - 1] < at[i]) {
            idle_time += at[i] - com[i - 1];
            com[i] = at[i] + bt[i];
```

```
        } else {
            com[i] = com[i - 1] + bt[i];
        }
        tat[i] = com[i] - at[i];
        wt[i] = tat[i] - bt[i];
    }

    for (i = 0; i < pn; i++)
        avg += tat[i];

    avgtat = avg / pn;

    for (i = 0; i < pn; i++)
        avg2 += wt[i];

    avgwt = avg2 / pn;

    printf("pno\tat\tbt\tcom\ttat\twt\t\n");
    for (i = 0; i < pn; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], com[i],
tat[i], wt[i]);
    }

    printf("AVGTAT= %f\nAWT= %f\nCPU Idle Time= %d\n", avgtat, avgwt,
idle_time);

    // Print Gantt Chart
    printf("\nGantt Chart:\n");
    printf("|0");
    for (i = 0; i < pn; i++) {
        printf("-P%d-%d", i+1, com[i]);
    }
    printf("|");

    return 0;
}
```

File name: SJF.c
(The following code simulates Shortest Job First process scheduling algorithm)

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

int main()
{
    int n,i,j;
```

```
    printf("\n----------------------------------------\n\t      SJF
Scheduling\n----------------------------------------\n");
    printf("Enter number of processes for SJF SCHEDULING: ");
    scanf("%d",&n);
    int Arr[n],BT[n];
    printf("\nArrival times: ");
    for(i=0;i<n;i++)
        scanf("%d",&Arr[i]);
    printf("\nBurst times: ");
    for(i=0;i<n;i++)
        scanf("%d",&BT[i]);
    int Bt[n],Ct[n],P[n];
    for(i=0;i<n;i++)
    {
        Bt[i]=BT[i];
        P[i]=i+1;  //storing process number
    }
    //int count=0;
    j=0;
    int current_time = 0; // Track current time
    char gantt_chart[1000] = "0"; // Initialize Gantt chart string
    while(j<n)
    {
        int min=1000,ind=-1;
        for(i=0;i<n;i++)
        {
            if(Bt[i]==min)
            {
                if(Arr[i]<Arr[ind])
                    ind=i;
            }

            else if(Bt[i]<min)
            {
                if(Arr[i]<=current_time)
                {
                    min=Bt[i];
                    ind=i;
                }
            }
        }
        if(ind!=-1)
        {
            Ct[ind]=BT[ind]+current_time;
            current_time+=BT[ind];
            j++;
            Bt[ind]=1001;
```

```
            char temp[50];
            sprintf(temp, " -> [P%d] -> %d", P[ind], current_time); //
Format new Gantt chart entry
            strcat(gantt_chart, temp); // Add new entry to Gantt chart
string
        }
        else
        {
            current_time++;
        }
    }

    int TAT[n],WT[n];

    for(i=0;i<n;i++)
        TAT[i]=Ct[i]-Arr[i];

    for(i=0;i<n;i++)
        WT[i]=TAT[i]-BT[i];

    printf("\n\nProcess Table:\nPid\tAT\tBT\tCT\tTAT\tWT\n");
    for(i=0;i<n;i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\n",(i+1),Arr[i],BT[i],Ct[i],TAT[i],WT[i]
);

    float sum=0;
    for(i=0;i<n;i++)
        sum+=(float)TAT[i];
    printf("Average TAT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
        sum+=(float)WT[i];
    printf("\nAverage WT: %f",sum/n);

    // Print Gantt Chart last
    printf("\n\nGantt Chart:\n%s|\n", gantt_chart);

    return 0;
}
```

File name: SRTF.c
(The following code simulates Shortest Remaining Time First process scheduling algorithm)

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

```
int main()
{
    printf("\n----------------------------------------\n\t     SRTF
Scheduling\n----------------------------------------\n");
    int n,i,j;
    printf("Enter number of processes for SRTF SCHEDULING: ");
    scanf("%d",&n);
    int Arr[n],BT[n];
    printf("Arrival times: ");
    for(i=0;i<n;i++){
        scanf("%d",&Arr[i]);
    }
    printf("\nBurst times: ");
    for(i=0;i<n;i++){
        scanf("%d",&BT[i]);
    }
    int Bt[n],Ct[n],Rt[n];
    for(i=0;i<n;i++)
    Bt[i]=BT[i];
    for(i=0;i<n;i++)
    Rt[i]=-1;
    int count=0,prev=-1;
    j=0;
    char gantt_chart[1000] = "0"; // Initialize Gantt chart string
    while(j<n)
    {
        int min=1000,ind=-1;
        for(i=0;i<n;i++)
        {
            if(Bt[i]==min && Bt[i]>0)
            {
                if(Arr[i]<Arr[ind])
                    ind=i;
            }

            else if(Bt[i]<min && Bt[i]>0)
            {
                if(Arr[i]<=count)
                {
                    min=Bt[i];
                    ind=i;
                }
            }

        }
```

```
        if(ind!=-1)
        {
            Bt[ind]-=1;
            char temp[50];
            sprintf(temp, " -> [P%d] -> %d", ind+1, count+1); // Format
new Gantt chart entry
            strcat(gantt_chart, temp); // Add new entry to Gantt chart
string
        }

        if(ind!=prev && Rt[ind]==-1)
        {
            Rt[ind]=count-Arr[ind];
        }

        count++;

        if(Bt[ind]==0 && ind!=-1)
        {
            Ct[ind]=count;
            j++;
        }
        prev=ind;
    }

    int TAT[n],WT[n];

    for(i=0;i<n;i++)
    TAT[i]=Ct[i]-Arr[i];

    for(i=0;i<n;i++)
    WT[i]=TAT[i]-BT[i];

    printf("\n\nProcess Table:\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for(i=0;i<n;i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",(i+1),Arr[i],BT[i],Ct[i],TAT[i],W
T[i],Rt[i]);

    float sum=0;
    for(i=0;i<n;i++)
    sum+=(float)TAT[i];
    printf("Average TAT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
    sum+=(float)WT[i];
    printf("\nAverage WT: %f",sum/n);
```

```
    sum=0;
    for(i=0;i<n;i++)
    sum+=(float)Rt[i];
    printf("\nAverage RT: %f",sum/n);

    // Print Gantt Chart last
    printf("\n\nGantt Chart:\n%s|\n", gantt_chart);
    return 0;
}
```

File name: RR.c
(The following code simulates Round Robin process scheduling algorithm)

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

int main()
{
    printf("\n----------------------------------------\n\t ROUND ROBIN
Scheduling\n----------------------------------------\n");
    int n,i,j,tq,len=0;
    printf("Enter number of processes for ROUND ROBIN :");
    scanf("%d",&n);
    int Arr[n],BT[n];
    printf("\nArrival times: ");
    for(i=0;i<n;i++)
    scanf("%d",&Arr[i]);
    printf("\nBurst times: ");
    for(i=0;i<n;i++)
    scanf("%d",&BT[i]);
    printf("\nEnter Time Quanta: ");
    scanf("%d",&tq);
    int Bt[n],Ct[n],Rt[n];
    for(i=0;i<n;i++)
    {
        Bt[i]=BT[i];
        len+=BT[i];
        Rt[i]=-1;
    }
    int ready[len];
    int count=0,front=0,rear=-1,dec=0;
    j=0;
    char gantt_chart[1000] = "0"; // Initialize Gantt chart string
    while(j<n)
    {
        int found=0;
        if(front>rear)
```

```
        {
            for(i=0;i<n;i++)
            if(Arr[i]==count)
            {
                ready[++rear]=i;
                found=1;
            }
            if(found==0)
            {
                if(dec==0)
                {
                    char temp[50];
                    sprintf(temp, "%d-Idle-", count); // Format new
Gantt chart entry
                    strcat(gantt_chart, temp); // Add new entry to Gantt
chart string
                    dec=1;
                }
                count++;
            }
        }
        else
        {
            int k,ind=ready[front++];

            char temp[50];
            sprintf(temp, "%d-P%d-", count, ind+1); // Format new Gantt
chart entry
            strcat(gantt_chart, temp); // Add new entry to Gantt chart
string

            if(Rt[ind]==-1)
            Rt[ind]=count-Arr[ind];

            if(Bt[ind]<=tq)
            {
                for(k=0;k<Bt[ind];k++)
                {
                    count++;
                    for(i=0;i<n;i++)
                    {
                        if(Arr[i]==count)
                        ready[++rear]=i;
                    }
                }
                Bt[ind]=0;
                Ct[ind]=count;
```

```
                j++;
            }
            else
            {
                for(k=0;k<tq;k++)
                {
                    count++;
                    for(i=0;i<n;i++)
                    {
                        if(Arr[i]==count)
                        ready[++rear]=i;
                    }
                }
                Bt[ind]-=tq;
                ready[++rear]=ind;
            }
        }
    }

    int TAT[n],WT[n];

    for(i=0;i<n;i++)
    TAT[i]=Ct[i]-Arr[i];

    for(i=0;i<n;i++)
    WT[i]=TAT[i]-BT[i];


    printf("\nProcess Table:\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for(i=0;i<n;i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",(i+1),Arr[i],BT[i],Ct[i],TAT[i],W
T[i],Rt[i]);

    float sum=0;
    for(i=0;i<n;i++)
    sum+=(float)TAT[i];
    printf("Average TAT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
    sum+=(float)WT[i];
    printf("\nAverage WT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
    sum+=(float)Rt[i];
    printf("\nAverage RT: %f",sum/n);
```

```
    // Print Gantt Chart last
    printf("\n\nGantt Chart:\n%s%d|\n", gantt_chart, count);

    return 0;
}
```

File name: PP.c
(The following code simulates Priority Preemptive process scheduling algorithm)

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

int main() {
    printf("\n---------------------------------------\n    PRIORITY
PREEMPTIVE Scheduling\n---------------------------------------\n");
    int n,i,j;
    printf("Enter number of processes for PRIORITY PREEMPTIVE
SCHEDULING: ");
    scanf("%d",&n);
    int Arr[n],BT[n],PR[n];
    printf("\nArrival times: ");
    for(i=0;i<n;i++)
        scanf("%d",&Arr[i]);
    printf("\nBurst times: ");
    for(i=0;i<n;i++)
        scanf("%d",&BT[i]);
    printf("\nPriority: ");
    for(i=0;i<n;i++)
        scanf("%d",&PR[i]);
    int Bt[n],Ct[n],Rt[n],Pr[n];
    for(i=0;i<n;i++)
        Bt[i]=BT[i];
    for(i=0;i<n;i++)
        Pr[i]=PR[i];
    for(i=0;i<n;i++)
        Rt[i]=-1;
    int count=0,prev=-1,dec=0;
    j=0;
    int choice;
    printf("\nOption:\n1.Lowest number Highest priority.\n2.Lowest
number Lowest priority.\n>>>");
    scanf("%d",&choice);

    // String to store the Gantt chart
    char ganttChart[1000] = "";
```

```
    switch(choice) {
        case 1:
            while(j<n) {
                int min=1000,ind=-1;
                for(i=0;i<n;i++) {
                    if(Pr[i]==min && Bt[i]>0) {
                        if(Arr[i]<Arr[ind])
                            ind=i;
                    } else if(Pr[i]<min && Bt[i]>0) {
                        if(Arr[i]<=count) {
                            min=Pr[i];
                            ind=i;
                        }
                    }
                }

                if(ind!=-1) {
                    Bt[ind]-=1;
                    if(ind!=prev)
                        sprintf(ganttChart + strlen(ganttChart),
"%d-P%d-",count,(ind+1));
                } else {
                    if(dec==0)
                        sprintf(ganttChart + strlen(ganttChart),
"%d-Idle-",count);
                    dec=1;
                }

                if(ind!=prev && Rt[ind]==-1)
                    Rt[ind]=count-Arr[ind];

                count++;

                if(Bt[ind]==0 && ind!=-1) {
                    Ct[ind]=count;
                    j++;
                    dec=0;
                }
                prev=ind;
            }
        break;

        case 2:
            while(j<n) {
                int max=-10,ind=-1;
                for(i=0;i<n;i++) {
                    if(Pr[i]==max && Bt[i]>0) {
```

```
                        if(Arr[i]<Arr[ind])
                            ind=i;
                    } else if(Pr[i]>max && Bt[i]>0) {
                        if(Arr[i]<=count) {
                            max=Pr[i];
                            ind=i;
                        }
                    }
                }

                if(ind!=-1) {
                    Bt[ind]-=1;
                    if(ind!=prev)
                        sprintf(ganttChart + strlen(ganttChart),
"%d-P%d-",count,(ind+1));
                } else {
                    if(dec==0)
                        sprintf(ganttChart + strlen(ganttChart),
"%d-Idle-",count);
                    dec=1;
                }


                if(ind!=prev && Rt[ind]==-1)
                    Rt[ind]=count-Arr[ind];

                count++;

                if(Bt[ind]==0 && ind!=-1) {
                    Ct[ind]=count;
                    j++;
                    dec=0;
                }
                prev=ind;
            }
        break;

        default:
            printf("Wrong choice..\n");
        break;
    }

    // Print the Gantt chart


    // Print other process information
    int TAT[n],WT[n];
```

```
    for(i=0;i<n;i++)
        TAT[i]=Ct[i]-Arr[i];

    for(i=0;i<n;i++)
        WT[i]=TAT[i]-BT[i];

    printf("\nProcess Table:\nPid\tPr\tAT\tBT\tCT\tTAT\tWT\n");
    for(i=0;i<n;i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",(i+1),PR[i],Arr[i],BT[i],Ct[i],TA
T[i],WT[i]);

    float sum=0;
    for(i=0;i<n;i++)
        sum+=(float)TAT[i];
    printf("Average TAT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
        sum+=(float)WT[i];
    printf("\nAverage WT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
        sum+=(float)Rt[i];
    printf("\nAverage RT: %f",sum/n);
    printf("\n\nGantt chart:\n|%s%d|\n", ganttChart, count);
    return 0;
}
```

File name: NPP.c
(The following code simulates Priority Non Preemptive process scheduling algorithm)

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

int main()
{
    int n,i,j;
    printf("\n----------------------------------------\n   PRIORITY NON
PREEMPTIVE Scheduling\n----------------------------------------\n");
    printf("Enter number of processes for NON PREEMPTIVE PRIORITY
SCHEDULING: ");
    scanf("%d",&n);
    int Arr[n],BT[n],PR[n];
    printf("\nArrival times: ");
    for(i=0;i<n;i++)
```

```
    scanf("%d",&Arr[i]);
    printf("\nBurst times: ");
    for(i=0;i<n;i++)
    scanf("%d",&BT[i]);
    printf("\nPriority: ");
    for(i=0;i<n;i++)
    scanf("%d",&PR[i]);
    int Pr[n],Ct[n];
    for(i=0;i<n;i++)
    Pr[i]=PR[i];
    int count=0,dec=0;
    j=0;
    int choice;
    printf("\nOption:\n1.Lowest number Highest priority.\n2.Lowest
number Lowest priority.\n>>>");
    scanf("%d",&choice);

    char gantt_chart[1000] = "0"; // Initialize Gantt chart string
    switch(choice)
    {
        case 1:
            while(j<n)
            {
                int min=1000,ind=-1;
                for(i=0;i<n;i++)
                {
                    if(Pr[i]==min)
                    {
                        if(Arr[i]<Arr[ind])
                        ind=i;
                    }

                    else if(Pr[i]<min)
                    {
                        if(Arr[i]<=count)
                        {
                            min=Pr[i];
                            ind=i;
                        }
                    }

                }
                if(ind!=-1)
                {
                    char temp[50];
                    sprintf(temp, "%d-P%d-", count, ind+1); // Format
new Gantt chart entry
```

```
                    strcat(gantt_chart, temp); // Add new entry to Gantt
chart string
                    Ct[ind]=BT[ind]+count;
                    count+=BT[ind];
                    j++;
                    Pr[ind]=1001;
                    dec=0;
                }
                else
                {
                    if(dec==0)
                    {
                        char temp[50];
                        sprintf(temp, "%d-Idle-", count); // Format new
Gantt chart entry
                        strcat(gantt_chart, temp); // Add new entry to
Gantt chart string
                        dec=1;
                    }
                    count++;
                }
            }
        break;
        case 2:
            while(j<n)
            {
                int max=-1,ind=-1;
                for(i=0;i<n;i++)
                {
                    if(Pr[i]==max)
                    {
                        if(Arr[i]<Arr[ind])
                        ind=i;
                    }

                    else if(Pr[i]>max)
                    {
                        if(Arr[i]<=count)
                        {
                            max=Pr[i];
                            ind=i;
                        }
                    }

                }
                if(ind!=-1)
                {
```

```
                    char temp[50];
                    sprintf(temp, "%d-P%d-", count, ind+1); // Format
new Gantt chart entry
                    strcat(gantt_chart, temp); // Add new entry to Gantt
chart string
                    Ct[ind]=BT[ind]+count;
                    count+=BT[ind];
                    j++;
                    Pr[ind]=-2;
                    dec=0;
                }
                else
                {
                    if(dec==0)
                    {
                        char temp[50];
                        sprintf(temp, "%d-Idle-", count); // Format new
Gantt chart entry
                        strcat(gantt_chart, temp); // Add new entry to
Gantt chart string
                        dec=1;
                    }
                    count++;
                }
            }
        break;
        default:
            printf("Wrong choice..\n");
        break;
    }

    int TAT[n],WT[n];

    for(i=0;i<n;i++)
    TAT[i]=Ct[i]-Arr[i];

    for(i=0;i<n;i++)
    WT[i]=TAT[i]-BT[i];

    printf("\nProcess Table:\nPid\tPr\tAT\tBT\tCT\tTAT\tWT\n");
    for(i=0;i<n;i++)

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",(i+1),PR[i],Arr[i],BT[i],Ct[i],TA
T[i],WT[i]);

    float sum=0;
    for(i=0;i<n;i++)
```

```
    sum+=(float)TAT[i];
    printf("Average TAT: %f",sum/n);
    sum=0;
    for(i=0;i<n;i++)
    sum+=(float)WT[i];
    printf("\nAverage WT: %f",sum/n);

    // Print Gantt Chart last
    printf("\n\nGantt Chart:\n|%s%d|\n", gantt_chart, count);

    return 0;
}
```

File name: simulate.c
(The following code simulates all the process scheduling algorithm)

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

int main() {
    int choice;
    char cont;

    do {
        // Display the menu

printf("------------------------------------------------------------\n");
        printf("Select the scheduling algorithm to simulate:\n");
        printf("1. First Come First Serve (FCFS)\n");
        printf("2. Non Pre-emptive Priority (NPP)\n");
        printf("3. Pre-emptive Priority (PP)\n");
        printf("4. Round Robin (RR)\n");
        printf("5. Shortest Job First (SJF)\n");
        printf("6. Shortest Remaining Time First (SRTF)\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        // Execute the selected algorithm
        switch (choice) {
            case 1:
                // Call the FCFS program
                system("FCFS.exe");
                break;
            case 2:
                // Call the NPP program
                system("NPP.exe");
```

```
                break;
            case 3:
                // Call the PP program
                system("PP.exe");
                break;
            case 4:
                // Call the RR program
                system("RR.exe");
                break;
            case 5:
                // Call the SJF program
                system("SJF.exe");
                break;
            case 6:
                // Call the SRTF program
                system("SRTF.exe");
                break;
            case 7:
                // Exit the program
                return 0;
            default:
                printf("Invalid choice\n");
                break;
        }

        // Ask if the user wants to continue
        printf("\nDo you want to continue? (y/n): ");
        scanf(" %c", &cont);
    } while (cont == 'y' || cont == 'Y');

    return 0;
}
```

Input and Output

```
PS C:\Users\ramee\Desktop\OS\output> & .\'simulate.exe'
------------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 1

---------------------------------------
             FCFS Scheduling
---------------------------------------

Enter the number of processes for FCFS SCHEDULING: 5

Enter the arrival times: 2 0 2 3 4

Enter the burst times: 2 1 3 5 4
pno     at      bt      com     tat     wt
P1      0       1       1       1       0
P2      2       2       4       2       0
P3      2       3       7       5       2
P4      3       5       12      9       4
P5      4       4       16      12      8
AVGTAT= 5.800000
AWT= 2.800000
CPU Idle Time= 1

Gantt Chart:
|0-P1-1-P2-4-P3-7-P4-12-P5-16|
Do you want to continue? (y/n): y
------------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 2
```

```
----------------------------------------
   PRIORITY NON PREEMPTIVE Scheduling
----------------------------------------
Enter number of processes for NON PREEMPTIVE PRIORITY SCHEDULING: 7

Arrival times: 0 1 3 4 5 6 10

Burst times: 8 2 4 1 6 5 1

Priority: 3 4 4 5 2 6 1

Option:
1.Lowest number Highest priority.
2.Lowest number Lowest priority.
>>>1

Process Table:
Pid     Pr      AT      BT      CT      TAT     WT
P1      3       0       8       8       8       0
P2      4       1       2       17      16      14
P3      4       3       4       21      18      14
P4      5       4       1       22      18      17
P5      2       5       6       14      9       3
P6      6       6       5       27      21      16
P7      1       10      1       15      5       4
Average TAT: 13.571428
Average WT: 9.714286

Gantt Chart:
|00-P1-8-P5-14-P7-15-P2-17-P3-21-P4-22-P6-27|

Do you want to continue? (y/n): y
------------------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 3

----------------------------------------
    PRIORITY PREEMPTIVE Scheduling
----------------------------------------
```

```
Enter number of processes for PRIORITY PREEMPTIVE SCHEDULING: 7

Arrival times: 0 1 3 4 5 6 10

Burst times: 8 2 4 1 6 5 1

Priority: 3 4 4 5 2 6 1

Option:
1.Lowest number Highest priority.
2.Lowest number Lowest priority.
>>>1

Process Table:
Pid     Pr      AT      BT      CT      TAT     WT
P1      3       0       8       15      15      7
P2      4       1       2       17      16      14
P3      4       3       4       21      18      14
P4      5       4       1       22      18      17
P5      2       5       6       12      7       1
P6      6       6       5       27      21      16
P7      1       10      1       11      1       0
Average TAT: 13.714286
Average WT: 9.857142
Average RT: 8.714286

Gantt chart:
|0-P1-5-P5-10-P7-11-P5-12-P1-15-P2-17-P3-21-P4-22-P6-27|

Do you want to continue? (y/n): y
-----------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 4

---------------------------------------
         ROUND ROBIN Scheduling
---------------------------------------
Enter number of processes for ROUND ROBIN :5

Arrival times: 0 5 1 6 8
```

```
Burst times: 8 2 7 3 5

Enter Time Quanta: 3

Process Table:
Pid     AT      BT      CT      TAT     WT      RT
P1      0       8       22      22      14      0
P2      5       2       11      6       4       4
P3      1       7       23      22      15      2
P4      6       3       14      8       5       5
P5      8       5       25      17      12      9
Average TAT: 15.000000
Average WT: 10.000000
Average RT: 4.000000
Gantt Chart:
00-P1-3-P3-6-P1-9-P2-11-P4-14-P3-17-P5-20-P1-22-P3-23-P5-25|

Do you want to continue? (y/n): y
-------------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 5

---------------------------------------
              SJF Scheduling
---------------------------------------
Enter number of processes for SJF SCHEDULING: 5

Arrival times: 2 1 4 0 2

Burst times: 1 5 1 6 3


Process Table:
Pid     AT      BT      CT      TAT     WT
P1      2       1       7       5       4
P2      1       5       16      15      10
P3      4       1       8       4       3
P4      0       6       6       6       0
P5      2       3       11      9       6
```

```
Average TAT: 7.800000
Average WT: 4.600000

Gantt Chart:
0 -> [P4] -> 6 -> [P1] -> 7 -> [P3] -> 8 -> [P5] -> 11 -> [P2] -> 16|

Do you want to continue? (y/n): y
------------------------------------------------------
Select the scheduling algorithm to simulate:
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 6

--------------------------------------
             SRTF Scheduling
--------------------------------------
Enter number of processes for SRTF SCHEDULING: 5
Arrival times: 2 1 4 0 2

Burst times: 1 5 1 6 3


Process Table:
Pid     AT      BT      CT      TAT     WT      RT
P1      2       1       3       1       0       0
P2      1       5       16      15      10      10
P3      4       1       5       1       0       0
P4      0       6       11      11      5       0
P5      2       3       7       5       2       1

Average TAT: 6.600000
Average WT: 3.400000Average RT: 2.200000

Gantt Chart:
0 -> [P4] -> 1 -> [P4] -> 2 -> [P1] -> 3 -> [P5] -> 4 -> [P3] -> 5 ->
[P5] -> 6 -> [P5] -> 7 -> [P4] -> 8 -> [P4] -> 9 -> [P4] -> 10 -> [P4]
-> 11 -> [P2] -> 12 -> [P2] -> 13 -> [P2] -> 14 -> [P2] -> 15 -> [P2] ->
16|

Do you want to continue? (y/n): y
------------------------------------------------------
Select the scheduling algorithm to simulate:
```

```
1. First Come First Serve (FCFS)
2. Non Pre-emptive Priority (NPP)
3. Pre-emptive Priority (PP)
4. Round Robin (RR)
5. Shortest Job First (SJF)
6. Shortest Remaining Time First (SRTF)
7. Exit
Enter your choice: 7
```

GitHub Repository Link