Jasmeet Singh -1BM22CS116

LAB-1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

→FCFS

```
#include<stdio.h>

void main()
{
   int n;
   printf("Enter number of processes:\n");
   scanf("%d",&n);
   int pr[n], at[n], bt[n], ct[n], tat[n], wt[n];
   printf("Enter Process number:\n");
   for (int i=0; i<n; i++)
   {
      scanf("%d", &pr[i]);
   }
   printf("Enter Arrival Time:\n");
   for (int i=0; i<n; i++)
   {
      scanf("%d", &at[i]);
   }
   printf("Enter Burst Time:\n");
   for (int i=0; i<n; i++)
   {
      scanf("%d", &bt[i]);
   }
   int temp1, temp2, temp3;
   for (int i=0; i<n; i++)
   {
      for (int j=i+1; j<n; j++)
      {
         if (at[j]<at[i])
         {
            temp1 = at[j];
            at[j] = at[i];
            at[i] = temp1;

            temp2 = bt[j];
            bt[j] = bt[i];
            bt[i] = temp2;

            temp3 = pr[j];
            pr[j] = pr[i];
            pr[i] = temp3;
         }
      }
   }
   int x=at[0];
   for (int i=0; i<n; i++)
   {
```

```
        if (x<at[i])
        {
            x = at[i];
        }
        ct[i] = bt[i] + x;
        x = ct[i];
    }
    for (int i=0; i<n; i++)
    {
        tat[i] = ct[i] - at[i];
    }
    for (int i=0; i<n; i++)
    {
        wt[i] = tat[i] - bt[i];
    }
    for (int i=0; i<n; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", pr[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    float avg_tat = 0, avg_wt = 0;
    for (int i=0; i<n; i++)
    {
        avg_tat = avg_tat + tat[i];
        avg_wt = avg_wt + wt[i];
    }
    avg_tat = avg_tat/n;
    avg_wt = avg_wt/n;
    printf("The average Turnaround time is: %f", avg_tat);
    printf("\nThe average Waiting time is: %f`", avg_wt);
}
```

```
Enter number of processes:
4
Enter Process number:
1
2
3
4
Enter Arrival Time:
0
1
5
6
Enter Burst Time:
2
2
3
4
1       0       2       2       2       0
2       1       2       4       3       1
3       5       3       8       3       0
4       6       4       12      6       2
The average Turnaround time is: 3.500000
The average Waiting time is: 0.750000`
```

→ SJF (pre-emptive)

```
#include<stdio.h>

// Function to find the waiting time, turnaround time, and completion time for all processes using SJF (Preemptive)
void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int ct[])
{
   int remaining[n]; // Array to store remaining burst time of processes
   int currentTime = 0; // Current time
   int completed = 0; // Counter for completed processes

   // Initialize remaining array with burst times
   for (int i = 0; i < n; i++)
      remaining[i] = bt[i];

   while (completed < n)
   {
      int shortest = -1; // Index of shortest remaining time process

      // Find process with shortest remaining burst time
      for (int i = 0; i < n; i++)
      {
         if (at[i] <= currentTime && remaining[i] > 0)
         {
            if (shortest == -1 || remaining[i] <= remaining[shortest])
               shortest = i;
         }
```

```
        }

        // If no process found, move to next time
        if (shortest == -1)
        {
            currentTime++;
            continue;
        }

        // Reduce remaining time of the process
        remaining[shortest]--;

        // If the process is completed
        if (remaining[shortest] == 0)
        {
            completed++;
            // Set completion time for the process
            ct[shortest] = currentTime + 1;
            // Calculate waiting time and turnaround time for the process
            wt[shortest] = ct[shortest] - bt[shortest] - at[shortest];
            tat[shortest] = ct[shortest] - at[shortest];
        }

        // Move to the next time
        currentTime++;
    }

    // Print the table
    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    float avg_tat = 0, avg_wt = 0;
    for (int i = 0; i < n; i++)
    {
        avg_tat += tat[i];
        avg_wt += wt[i];
    }
    avg_tat /= n;
    avg_wt /= n;
    printf("The average Turnaround time is %f\n", avg_tat);
    printf("The average Waiting time is %f\n", avg_wt);
}

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

    // Process id's
    int processes[n];
    // Burst time of all processes
    int burst_time[n];
    // Arrival time of all processes
    int arrival_time[n];
```

```
    printf("Enter Process Number:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &processes[i]);
    }
    printf("Enter Arrival Time:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arrival_time[i]);
    }
    printf("Enter Burst Time:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &burst_time[i]);
    }

    // Arrays to store waiting time, turnaround time, and completion time
    int wt[n], tat[n], ct[n];

    printf("\nSJF (Preemptive) Scheduling:\n");
    findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, ct);
}
```

```
Enter the number of processes: 5
Enter Process Number:
1
2
3
4
5
Enter Arrival Time:
2
1
4
0
2
Enter Burst Time:
1
5
1
6
3

SJF (Preemptive) Scheduling:
1               2               1               3               1               0
2               1               5               16              15              10
3               4               1               5               1               0
4               0               6               11              11              5
5               2               3               7               5               2
The average Turnaround time is 6.600000
The average Waiting time is 3.400000
```

→ SJF (non-pre-emptive)

```
#include<stdio.h>

// Function to find the waiting time, turnaround time, response time, and completion time for all processes using SJF (Non-preemptive)
void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int rt[], int ct[])
```

```
{
    int completion[n]; // Array to store completion times of processes
    int remaining[n]; // Array to store remaining burst time of processes

    // Initialize remaining array with burst times
    for (int i = 0; i < n; i++)
        remaining[i] = bt[i];

    int currentTime = 0; // Current time

    // Find process with shortest burst time
    for (int i = 0; i < n; i++)
    {
        int shortest = -1;
        for (int j = 0; j < n; j++)
        {
            if (at[j] <= currentTime && remaining[j] > 0)
            {
                if (shortest == -1 || remaining[j] < remaining[shortest])
                    shortest = j;
            }
        }

        if (shortest == -1)
        {
            currentTime++;
            continue;
        }

        // Set completion time for the process
        completion[shortest] = currentTime + remaining[shortest];
        // Update current time
        currentTime = completion[shortest];
        // Calculate waiting time, turnaround time, and response time for the process
        wt[shortest] = currentTime - bt[shortest] - at[shortest];
        tat[shortest] = currentTime - at[shortest];
        rt[shortest] = wt[shortest]; // Response time for non-preemptive SJF is the same as waiting time
        // Mark the process as completed
        remaining[shortest] = 0;
    }

    // Copy completion times to ct[] and print the table
    for (int i = 0; i < n; i++)
    {
        ct[i] = completion[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], at[i], bt[i], ct[i], wt[i], tat[i], rt[i]);
    }
    float avg_tat = 0, avg_wt = 0;
    for (int i = 0; i < n; i++)
    {
        avg_tat += tat[i];
        avg_wt += wt[i];
    }
    avg_tat /= n;
    avg_wt /= n;
    printf("The average Turnaround time is %f\n", avg_tat);
    printf("The average Waiting time is %f\n", avg_wt);
```

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

   // Process id's
   int processes[n];
   // Burst time of all processes
   int burst_time[n];
   // Arrival time of all processes
   int arrival_time[n];
   printf("Enter Process Number:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &processes[i]);
   }
   printf("Enter Arrival Time:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arrival_time[i]);
   }
   printf("Enter Burst Time:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &burst_time[i]);
   }
   // Arrays to store waiting time, turnaround time, response time, and completion time
   int wt[n], tat[n], rt[n], ct[n];

   // Initialize response times to -1
   for (int i = 0; i < n; i++)
      rt[i] = -1;
   printf("\nSJF (Non-preemptive) Scheduling:\n");
   findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, rt, ct);
}
```

```
Enter the number of processes: 5
Enter Process Number:
1
2
3
4
5
Enter Arrival Time:
2
1
4
0
2
Enter Burst Time:
1
5
1
6
3

SJF (Non-preemptive) Scheduling:
1               2               1               7               4               5               4
2               1               5               16              10              15              10
3               4               1               8               3               4               3
4               0               6               6               0               6               0
5               2               3               11              6               9               6
The average Turnaround time is 7.800000
The average Waiting time is 4.600000
```

LAB-2

Write a C program to simulate the following CPU scheduling algorithm to find

turnaround time and waiting time.

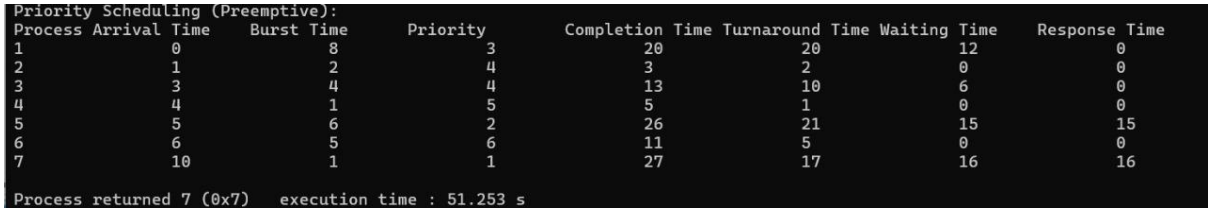
→ Priority (pre-emptive & non-pre-emptive)

```
#include <stdio.h>
#include <stdbool.h>
// Function to find the waiting time, turnaround time, and completion time for all processes using Priority Scheduling (Preemptive)
void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int ct[], int rt[], int priority[], bool isLowerPriorityHigher)
{
   int remaining[n]; // Array to store remaining burst time of processes
   int currentTime = 0; // Current time
   int completed = 0; // Counter for completed processes
   bool isFinished[n]; // Array to indicate if the process is finished
   // Initialize remaining array with burst times and set response times
   for (int i = 0; i < n; i++) {
      remaining[i] = bt[i];
      isFinished[i] = false;
      rt[i] = -1; // Response time is initially unset
   }
   while (completed < n) {
      int highestPriorityIndex = -1;
      int highestPriority = isLowerPriorityHigher ? 1000000 : -1; // Adjust initial value based on priority type

      // Find the process with the highest priority that has arrived and is not finished
      for (int i = 0; i < n; i++) {
         if (at[i] <= currentTime && !isFinished[i] &&
            ((isLowerPriorityHigher && priority[i] < highestPriority) ||
             (!isLowerPriorityHigher && priority[i] > highestPriority))) {
            highestPriority = priority[i];
            highestPriorityIndex = i;
```

```
        }      }
      // If no process is found, move to the next time
      if (highestPriorityIndex == -1) {
         currentTime++;
         continue;
      }
      int currentProcess = highestPriorityIndex;

      // Set response time if it's the first time the process is executed
      if (rt[currentProcess] == -1) {
         rt[currentProcess] = currentTime - at[currentProcess];
      }
      // Execute the process for 1 unit of time
      remaining[currentProcess]--;
      currentTime++;
      // If the process is completed
      if (remaining[currentProcess] == 0) {
         isFinished[currentProcess] = true;
         completed++;
         ct[currentProcess] = currentTime; // Set completion time for the process
         tat[currentProcess] = ct[currentProcess] - at[currentProcess]; // Calculate turnaround time
         wt[currentProcess] = tat[currentProcess] - bt[currentProcess]; // Calculate waiting time
      }   }
   // Print the table
   printf("Process\tArrival Time\tBurst Time\tPriority\tCompletion Time\tTurnaround Time\tWaiting
Time\tResponse Time\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
            processes[i], at[i], bt[i], priority[i], ct[i], tat[i], wt[i], rt[i]);
   }}
void main()
{
   // Number of processes
```

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

// Process id's
int processes[n];
// Burst time of all processes
int burst_time[n];
// Arrival time of all processes
int arrival_time[n];
// Priority of all processes
int priority[n];
// Priority type (true for lower number = higher priority, false for higher number = higher priority)
int priorityType;
bool isLowerPriorityHigher;
printf("Enter 1 if lower number indicates higher priority, 0 if higher number indicates higher priority: ");
scanf("%d", &priorityType);
isLowerPriorityHigher = (priorityType == 1);

printf("Enter Process Number:\n");
for (int i = 0; i < n; i++) {
   scanf("%d", &processes[i]);
}
printf("Enter Priority:\n");
for (int i = 0; i < n; i++) {
   scanf("%d", &priority[i]);
}
printf("Enter Arrival Time:\n");
for (int i = 0; i < n; i++) {
   scanf("%d", &arrival_time[i]);
}
printf("Enter Burst Time:\n");
for (int i = 0; i < n; i++) {
```

```
    scanf("%d", &burst_time[i]);
  }
  // Arrays to store waiting time, turnaround time, completion time, and response time
  int wt[n], tat[n], ct[n], rt[n];
  printf("\nPriority Scheduling (Preemptive):\n");
  findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, ct, rt, priority, isLowerPriorityHigher);
}
```



```
Priority Scheduling (Preemptive):
Process Arrival Time    Burst Time      Priority        Completion Time Turnaround Time Waiting Time    Response Time
1               0               8               3               20              20              12              0
2               1               2               4               3               2               0               0
3               3               4               4               13              10              6               0
4               4               1               5               5               1               0               0
5               5               6               2               26              21              15              15
6               6               5               6               11              5               0               0
7               10              1               1               27              17              16              16

Process returned 7 (0x7)   execution time : 51.253 s
```

→Round Robin (Experiment with different quantum sizes for RR algorithm)

```
#include <stdio.h>
#include <stdbool.h>
void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int ct[], int rt[], int quantum)
{
  int remaining[n]; // Array to store remaining burst time of processes
  bool firstResponse[n]; // Array to track if response time has been set
  int currentTime = 0; // Current time
  int completed = 0; // Counter for completed processes
  // Initialize remaining array with burst times and first response array
  for (int i = 0; i < n; i++) {
    remaining[i] = bt[i];
    firstResponse[i] = true;
  }
  // Queue to hold the indices of the processes
  int queue[n];
  int front = -1, rear = -1;
  // Function to add process to the queue
  void enqueue(int process) {
    if (rear == n - 1)
```

```
        rear = -1;
    queue[++rear] = process;
    if (front == -1)
        front = 0;}
// Function to remove process from the queue
int dequeue() {
    int process = queue[front];
    if (front == rear)
        front = rear = -1;
    else {
        front++;
        if (front == n)
            front = 0;
    }       return process;
}
// To track which processes have been added to the queue
bool inQueue[n];
for (int i = 0; i < n; i++)
    inQueue[i] = false;
while (completed < n) {
    // Add all processes to the queue that have arrived by the current time
    for (int i = 0; i < n; i++) {
        if (at[i] <= currentTime && !inQueue[i]) {
            enqueue(i);
            inQueue[i] = true;
        }       }
    // If no process is ready, increment the current time
    if (front == -1) {
        currentTime++;
        continue;       }
    int currentProcess = dequeue();
    // Set response time if it's the first time the process is executed
    if (firstResponse[currentProcess]) {
```

```
            rt[currentProcess] = currentTime - at[currentProcess];
            firstResponse[currentProcess] = false;
        }
        // Execute the process for the time quantum or until completion
        if (remaining[currentProcess] > quantum) {
            remaining[currentProcess] -= quantum;
            currentTime += quantum;
        } else {
            currentTime += remaining[currentProcess];
            remaining[currentProcess] = 0;
            completed++;
            // Set completion time for the process
            ct[currentProcess] = currentTime;
            // Calculate waiting time and turnaround time for the process
            tat[currentProcess] = ct[currentProcess] - at[currentProcess];
            wt[currentProcess] = tat[currentProcess] - bt[currentProcess];
        }
        // Add all processes to the queue that have arrived by the current time
        for (int i = 0; i < n; i++) {
            if (at[i] <= currentTime && !inQueue[i]) {
                enqueue(i);
                inQueue[i] = true;
            }       }
        // Re-enqueue the current process if it is not finished
        if (remaining[currentProcess] > 0) {
            enqueue(currentProcess);
        }    }
    // Print the table
    printf("Process\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\tResponse
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
                processes[i], at[i], bt[i], ct[i], tat[i], wt[i], rt[i]);
```

```
    }}
void main()
{
    // Number of processes
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    // Process id's
    int processes[n];
    // Burst time of all processes
    int burst_time[n];
    // Arrival time of all processes
    int arrival_time[n];
    printf("Enter Process Number:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i]);
    }
    printf("Enter Arrival Time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arrival_time[i]);
    }
    printf("Enter Burst Time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &burst_time[i]);
    }

    // Time quantum for Round Robin
    int quantum;
    printf("Enter the time quantum: ");
    scanf("%d", &quantum);
    // Arrays to store waiting time, turnaround time, completion time, and response time
    int wt[n], tat[n], ct[n], rt[n];
    printf("\nRound Robin Scheduling:\n");
```

```
    findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, ct, rt, quantum);
}
```

```
Round Robin Scheduling:
Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time    Response Time
1               0               5               13              13              8               0
2               1               3               12              11              8               1
3               2               1               5               3               2               2
4               3               2               9               6               4               4
5               4               3               14              10              7               5
```