Assignment No 3

Roll no: 33229

SJF:

```
#include <iostream>
#include <stdlib.h>
#include <vector>
using namespace std;
void findCT(int **copy, int **arr, int row, int col)
{
   int *CT = (int *)malloc(row * sizeof(int));
   int *TAT = (int *)malloc(row * sizeof(int));
   int *WT = (int *)malloc(row * sizeof(int));
   int completion_time = 0;
   vector<pair<int, pair<int, int>>> ganttChart; // To store the process execution order
   for (int i = 0; i < row; i++)
   {
      int start_time = completion_time;
      completion_time += arr[i][1];
      CT[i] = completion_time;
      ganttChart.push_back({arr[i][0], {start_time, completion_time}});
   }
   int *original_CT = (int *)malloc(row * sizeof(int));
   for (int i = 0; i < row; i++)
   {
      for (int j = 0; j < row; j++)
      {
         if (copy[i][0] == arr[j][0] && copy[i][1] == arr[j][1])
         {
            original_CT[i] = CT[j];
            break;
         }
      }
   }
   printf("AT\tBT\tCT\tTAT\tWT\n");
   for (int i = 0; i < row; i++)
   {
      int AT = copy[i][0];
      int BT = copy[i][1];
      int CT_val = original_CT[i];
      int TAT_val = CT_val - AT;
      int WT_val = TAT_val - BT;
      TAT[i] = TAT_val;
      WT[i] = WT_val;
      printf("%d\t%d\t%d\t%d\t%d\n", AT, BT, CT_val, TAT_val, WT_val);
   }
   cout << "\nGantt Chart:\n";
   for (const auto &entry : ganttChart)
   {
      cout << "| " << "P(" << entry.first << ") " << " | ";
   }
   cout << "\n";
   for (const auto &entry : ganttChart)
   {
      cout << entry.second.first << "\t\t";
   }
```

```
    cout << completion_time << "\n";

    free(CT);
    free(TAT);
    free(WT);
    free(original_CT);
}
void myfun(int **arr, int row, int col)
{
    int i, j;
    int temp;
    int **copy = (int **)malloc(row * sizeof(int *));
    for (i = 0; i < row; i++)
    {
        copy[i] = (int *)malloc(col * sizeof(int));
        for (j = 0; j < col; j++)
        {
            copy[i][j] = arr[i][j];
        }
    }
    for (j = 1; j < row - 1; j++)
    {
        for (i = j + 1; i < row; i++)
        {
            if (arr[j][1] > arr[i][1])
            {
                for (int k = 0; k < col; k++)
                {
                    temp = arr[j][k];
                    arr[j][k] = arr[i][k];
                    arr[i][k] = temp;
                }
            }
        }
    }
    findCT(copy, arr, row, col);

    for (i = 0; i < row; i++)
    {
        free(copy[i]);
    }
    free(copy);
}

int main()
{
    int i, j;
    int row, col;
    printf("Enter the number of rows: ");
    scanf("%d", &row);
    printf("Enter the number of columns: ");
    scanf("%d", &col);
    int **arr = (int **)malloc(row * sizeof(int *));
    for (i = 0; i < row; i++)
    {
        arr[i] = (int *)malloc(col * sizeof(int));
    }
    printf("Enter the elements of the array (AT BT):\n");
    for (i = 0; i < row; i++)
```

```
    {
        for (j = 0; j < col; j++)
        {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &arr[i][j]);
        }
        printf("\n");
    }
    printf("The elements of the array are:\n");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    myfun(arr, row, col);
    for (i = 0; i < row; i++)
    {
        free(arr[i]);
    }
    free(arr);
    return 0;
}
```

Output:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the array (AT BT):
Element [0][0]: 2
Element [0][1]: 6
Element [0][2]: 4

Element [1][0]: 8
Element [1][1]: 4
Element [1][2]: 8

Element [2][0]: 4
Element [2][1]: 9
Element [2][2]: 5

The elements of the array are:
2 6 4
8 4 8
4 9 5
AT     BT     CT     TAT     WT
2      6      6      4      -2
8      4      10     2      -2
4      9      19     15     6

Gantt Chart:
------------------------------
|   P1   |  P2  |   P3   |
------------------------------
0        6         10       19
```

Round Robin:

```
#include <stdio.h>
#include <stdlib.h>
struct Process
{
   int pid;
   int at;
   int bt;
   int rt;
   int wt;
   int tat;
};
struct Node
{
   struct Process *data;
   struct Node *next;
};
struct Queue
{
   struct Node *front;
   struct Node *rear;
   int size;
};
struct Process* create_process(int pid, int at, int bt)
{
   struct Process *p = (struct Process*)malloc(sizeof(struct Process));
   p->pid = pid;
   p->at = at;
   p->bt = bt;
   p->rt = bt;
   p->wt = 0;
   p->tat = 0;
   return p;
}
struct Node* create_node(struct Process *p)
{
   struct Node *node = (struct Node*)malloc(sizeof(struct Node));
   node->data = p;
   node->next = NULL;
   return node;
}
void init_queue(struct Queue *q)
{
   q->front = q->rear = NULL;
   q->size = 0;
}
int is_empty(struct Queue *q)
{
   return q->size == 0;
}
void enqueue(struct Queue *q, struct Process *p) {

   struct Node *node = create_node(p);
   if (is_empty(q))
    {
      q->front = q->rear = node;
```

```
    }
    else
    {
        q->rear->next = node;
        q->rear = node;
    }
    q->size++;
}
struct Process* dequeue(struct Queue *q)
{
    if (is_empty(q))
    {
        return NULL;
    }
    struct Node *temp = q->front;
    struct Process *p = temp->data;
    q->front = q->front->next;
    if (q->front == NULL)
    {
        q->rear = NULL;
    }
    free(temp);
    q->size--;
    return p;
}
void store_gantt_chart(int gantt_chart[][2], int *gantt_index, int time, int pid)
{
    gantt_chart[*gantt_index][0] = time;
    gantt_chart[*gantt_index][1] = pid;
    (*gantt_index)++;
}

void print_gantt_chart(int gantt_chart[][2], int gantt_index)
{
    printf("\nGantt Chart:\n");
    printf(" ");
    for (int i = 0; i < gantt_index; i++)
    {
        printf("--------");
    }
    printf("\n|");
    for (int i = 0; i < gantt_index; i++)
     {
        printf(" P%d |", gantt_chart[i][1]);
    }
    printf("\n ");
    for (int i = 0; i < gantt_index; i++)
    {
        printf("--------");
    }
    printf("\n");
    printf("0");
    for (int i = 0; i < gantt_index; i++)
    {
        printf("     %d", gantt_chart[i][0]);
    }
    printf("\n");
}
int main()
```

```
{
    int NOP, quant, i, time = 0, total_wt = 0, total_tat = 0;
    printf("Total number of processes in the system: ");
    scanf("%d", &NOP);
    struct Process *processes[NOP];
    for (i = 0; i < NOP; i++) {
        int at, bt;
        printf("\nEnter the Arrival and Burst time of Process[%d]\n", i + 1);
        printf("Arrival time: ");
        scanf("%d", &at);
        printf("Burst time: ");
        scanf("%d", &bt);
        processes[i] = create_process(i + 1, at, bt);
    }
    printf("Enter the Time Quantum for the process: ");
    scanf("%d", &quant);
    struct Queue q;
    init_queue(&q);
    for (i = 0; i < NOP; i++)
    {
        if (processes[i]->at == 0)
        {
            enqueue(&q, processes[i]);
        }
    }
    int gantt_chart[100][2];
    int gantt_index = 0;
    printf("\nProcess No\tBurst Time\tTAT\tWaiting Time\n");
    while (!is_empty(&q))
    {
        struct Process *current = dequeue(&q);
        if (current->rt > quant)
        {
            time += quant;
            current->rt -= quant;
            store_gantt_chart(gantt_chart, &gantt_index, time, current->pid);
        }
        else
        {
            time += current->rt;
            current->rt = 0;
            current->tat = time - current->at;
            current->wt = current->tat - current->bt;
            printf("Process No[%d]\t%d\t\t%d\t%d\n", current->pid, current->bt, current->tat, current->wt);
            store_gantt_chart(gantt_chart, &gantt_index, time, current->pid);
            total_wt += current->wt;
            total_tat += current->tat;
        }
        for (i = 0; i < NOP; i++)
        {
            if (processes[i]->at <= time && processes[i]->rt > 0 && processes[i] != current)
            {
                int already_in_queue = 0;
                struct Node *node = q.front;
                while (node != NULL) {
                    if (node->data == processes[i])
                    {
                        already_in_queue = 1;
                        break;
```

```
            }
            node = node->next;
          }
          if (!already_in_queue)
          {
            enqueue(&q, processes[i]);
          }
        }
      }
      if (current->rt > 0)
      {
        enqueue(&q, current);
      }
    }
    print_gantt_chart(gantt_chart, gantt_index);
    float avg_wt = (float)total_wt / NOP;
    float avg_tat = (float)total_tat / NOP;
    printf("\nAverage Turn Around Time: %.2f", avg_tat);
    printf("\nAverage Waiting Time: %.2f", avg_wt);

    return 0;
}
```

Output:

```
Total number of processes in the system: 4
Enter the Arrival and Burst time of Process[1]
Arrival time: 5
Burst time: 8

Enter the Arrival and Burst time of Process[2]
Arrival time: 2
Burst time: 3

Enter the Arrival and Burst time of Process[3]
Arrival time: 6
Burst time: 1

Enter the Arrival and Burst time of Process[4]
Arrival time: 6
Burst time: 3
Enter the Time Quantum for the process: 15

Process No      Burst Time      TAT     Waiting Time
Process No[2]   3               3       0
Process No[1]   8               8       0
Process No[3]   1               8       7
Process No[4]   3               11      8

Gantt Chart:
 ----------------------------
| P2 | P1 | P3 | P4 |
 ----------------------------
0     5     13     14     17

Average Turn Around Time: 7.50
Average Waiting Time: 3.75
```

preemtive SJF :

```
#include <stdio.h>
#include <limits.h>
void findWaitingTime(int proc[][3], int n, int wt[], int gantt_chart[][2], int *gantt_size)
{
   int rt[n];
   for (int i = 0; i < n; i++)
      rt[i] = proc[i][1];
   int complete = 0, t = 0, minm = INT_MAX;
   int shortest = 0, finish_time;
   int check = 0;
   while (complete != n)
   {
      minm = INT_MAX;
      check = 0;
      for (int j = 0; j < n; j++)
      {
         if ((proc[j][2] <= t) && (rt[j] < minm) && rt[j] > 0)
         {
            minm = rt[j];
            shortest = j;
            check = 1;
         }
      }
      if (check == 0)
      {
         t++;
         continue;
      }
      gantt_chart[*gantt_size][0] = t;
      gantt_chart[*gantt_size][1] = proc[shortest][0];
      (*gantt_size)++;
      rt[shortest]--;
      minm = rt[shortest];
      if (minm == 0)
         minm = INT_MAX;
      if (rt[shortest] == 0)
      {
         complete++;
         check = 0;
         finish_time = t + 1;
         wt[shortest] = finish_time - proc[shortest][1] - proc[shortest][2];
         if (wt[shortest] < 0)
            wt[shortest] = 0;
      }
      t++;
   }
}
void findTurnAroundTime(int proc[][3], int n, int wt[], int tat[])
{
   for (int i = 0; i < n; i++)
      tat[i] = proc[i][1] + wt[i];
}
void findavgTime(int proc[][3], int n)
{
   int wt[n], tat[n], total_wt = 0, total_tat = 0;
```

```
    int gantt_chart[100][2]; // Assuming max 100 time units in Gantt chart
    int gantt_size = 0;

    findWaitingTime(proc, n, wt, gantt_chart, &gantt_size);
    findTurnAroundTime(proc, n, wt, tat);
    printf("P\t\tBT\t\tWT\t\tTAT\n");
    for (int i = 0; i < n; i++)
    {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("%d\t\t%d\t\t%d\t\t%d\n", proc[i][0], proc[i][1], wt[i], tat[i]);
    }
    printf("\nAverage waiting time = %.2f\n", (float)total_wt / n);
    printf("Average turn around time = %.2f\n", (float)total_tat / n);
    printf("\nGantt Chart:\n ");
    for (int i = 0; i < gantt_size; i++)
    {
        printf("-------");
    }
    printf("\n|");
    for (int i = 0; i < gantt_size; i++)
    {
        printf(" P%d |", gantt_chart[i][1]);
    }
    printf("\n ");
    for (int i = 0; i < gantt_size; i++)
    {
        printf("-------");
    }
    printf("\n0");
    for (int i = 0; i < gantt_size; i++)
    {
        printf("    %d", gantt_chart[i][0] + 1);
    }
    printf("\n");
}
int main()
{
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int proc[n][3];
    for (int i = 0; i < n; i++)
    {
        printf("Enter Process ID, Burst Time, Arrival Time for process %d: ", i + 1);
        scanf("%d %d %d", &proc[i][0], &proc[i][1], &proc[i][2]);
    }
    findavgTime(proc, n);
    return 0;
}
```

Output:

```
Enter the number of processes: 3
Enter Process ID, Burst Time, Arrival Time for process 1: 1 7 6
Enter Process ID, Burst Time, Arrival Time for process 2: 2 6 3
Enter Process ID, Burst Time, Arrival Time for process 3: 3 9 7
P           BT          WT          TAT
1           7           3           10
```

```
2          6          0          6
3          9          9          18

Average waiting time = 4.00
Average turn around time = 11.33

Gantt Chart:
 ---------------------------------------------------------------------------------------------------------------------------
| P2 | P2 | P2 | P2 | P2 | P2 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P3 | P3 | P3 | P3 | P3 | P3 | P3 | P3 | P3 |
 ---------------------------------------------------------------------------------------------------------------------------
0   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
```

FCFS:

```
#include <stdio.h>
void fcfs(int arr[][4], int n)
{
   int i, j, temp;
   int current_time = 0;
   int total_wt = 0, total_tat = 0;
   float avg_wt, avg_tat;
   for (i = 0; i < n - 1; i++)
   {
      for (j = 0; j < n - i - 1; j++)
      {
         if (arr[j][0] > arr[j + 1][0])
         {
            temp = arr[j][0];
            arr[j][0] = arr[j + 1][0];
            arr[j + 1][0] = temp;
            temp = arr[j][1];
            arr[j][1] = arr[j + 1][1];
            arr[j + 1][1] = temp;
         }
      }
   }
   int gantt_chart[100][2]; // To store Gantt chart data
   int gantt_index = 0;

   for (i = 0; i < n; i++)
   {
      if (current_time < arr[i][0])
      {
         current_time = arr[i][0];
      }
      gantt_chart[gantt_index][0] = current_time; // Start time
      gantt_chart[gantt_index][1] = i + 1;       // Process ID
      gantt_index++;
      current_time += arr[i][1];
      arr[i][2] = current_time - arr[i][0]; // Turnaround time
      arr[i][3] = arr[i][2] - arr[i][1];    // Waiting time
      total_wt += arr[i][3];
      total_tat += arr[i][2];
   }
   avg_wt = (float)total_wt / n;
   avg_tat = (float)total_tat / n;
   printf("\nProcess\t Arrival Time \tBurst Time \tTurnaround Time \tWaiting Time\n");
   for (i = 0; i < n; i++)
   {
```

```
        printf("%d\t\t %d\t\t %d\t\t %d\t\t\t %d\n", i + 1, arr[i][0], arr[i][1], arr[i][2], arr[i][3]);
    }
    printf("\nAverage Waiting Time = %f", avg_wt);
    printf("\nAverage Turnaround Time = %f", avg_tat);
    printf("\nGantt Chart:\n ");
    for (i = 0; i < gantt_index; i++)
    {
        printf("-------");
    }
    printf("\n|");
    for (i = 0; i < gantt_index; i++)
    {
        printf(" P%d |", gantt_chart[i][1]);
    }
    printf("\n ");
    for (i = 0; i < gantt_index; i++)
    {
        printf("-------");
    }
    printf("\n0");
    for (i = 0; i < gantt_index; i++)
    {
        printf("     %d", gantt_chart[i][0]);
    }
    printf("     %d\n", current_time); // Printing the final time
}

int main()
{
    int n, i;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int arr[n][4];
    printf("\nEnter Arrival Time and Burst Time:\n");
    for (i = 0; i < n; i++)
    {
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &arr[i][0]);
        printf("Burst Time: ");
        scanf("%d", &arr[i][1]);
    }
    fcfs(arr, n);
    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter Arrival Time and Burst Time:
Process 1:
Arrival Time: 2
Burst Time: 7
Process 2:
Arrival Time: 6
Burst Time: 3
Process 3:
Arrival Time: 9
```

Burst Time: 4
Process 4:
Arrival Time: 6
Burst Time: 5

Process	Arrival Time	Burst Time	Turnaround Time	Waiting Time
1	2	7	7	0
2	6	3	6	3
3	6	5	11	6
4	9	4	12	8

Average Waiting Time = 4.250000
Average Turnaround Time = 9.000000
Gantt Chart:

```
 -------------------------
| P1 | P2 | P3 | P4 |
 -------------------------
0    2    9    12    17
```