Assignment No 3:

Name: Samruddhi Devram Khilari

Branch & Year: AIML-B SY

Prn: 12420020

Roll no: 2

Sub: Operating System Lab

#CPU Scheduling Algorithms.

1. **Q] First Come First Serve CPU Scheduling Algorithm.**

Open ▾ FCFS_process_scheduling.c ~/Desktop/OS Save

FCFS_process_scheduling.c | bankers_alog.c

```c
#include<stdio.h>

void findWaitingTime(int processes[], int n,
                          int bt[], int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;

    // calculating waiting time
    for (int  i = 1; i < n ; i++ )
        wt[i] =  bt[i-1] + wt[i-1] ;
}

void findTurnAroundTime( int processes[], int n,
                  int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int  i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

void findavgTime( int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt);

    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes   Burst time   Waiting time   Turn around time\n");

```

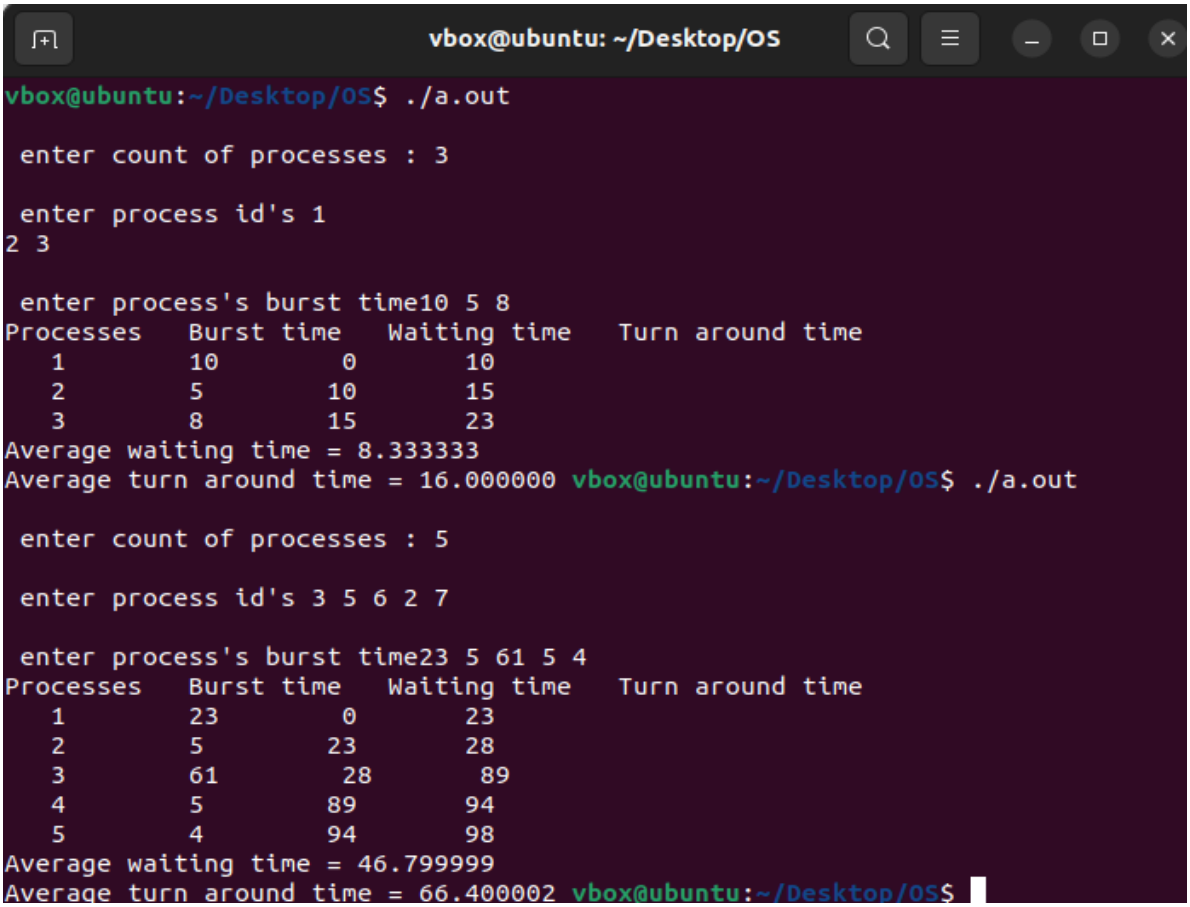
C ▾ Tab Width: 8 ▾ Ln 57, Col 5 ▾ INS

Open ▾ *FCFS_process_scheduling.c ~/Desktop/OS Save

*FCFS_process_scheduling.c | bankers_alog.c

```c
        printf("       %d ", bt[i] );
        printf("       %d",wt[i] );
        printf("       %d\n",tat[i] );
    }
    float s=(float)total_wt / (float)n;
    float t=(float)total_tat / (float)n;
    printf("Average waiting time = %f",s);
    printf("\n");
    printf("Average turn around time = %f ",t);
}
int main()
{
int n;
    printf("\n enter count of processes : ");
    scanf("%d",&n);
    //process id's
    int processes[n];
    //Burst time of all processes
    int  burst_time[n];
    printf("\n enter process id's ");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&processes[i]);
    }
    printf("\n enter process's burst time");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&burst_time[i]);
    }
    findavgTime(processes, n,  burst_time);
    return 0;
}
```

C ▾ Tab Width: 8 ▾ Ln 47, Col 3 ▾ INS



2. 1Q] Priority Preemptive CPU Scheduling Algorithm.

```
#include <stdio.h>

struct Process {
   int id;       // Process ID
   int burst;    // Burst time
   int priority;  // Priority
};

void sortProcesses(struct Process proc[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         // Sort by priority (lower number = higher priority)
         if (proc[j].priority > proc[j + 1].priority) {
            struct Process temp = proc[j];
            proc[j] = proc[j + 1];
            proc[j + 1] = temp;
         }
      }
   }
}

void priorityScheduling(struct Process proc[], int n) {
   sortProcesses(proc, n);

   int waitingTime[n];
   int turnaroundTime[n];

   // Calculate waiting time
   waitingTime[0] = 0; // First process has no waiting time
```

```
    for (int i = 1; i < n; i++) {
        waitingTime[i] = waitingTime[i - 1] + proc[i - 1].burst;
    }

    // Calculate turnaround time
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = waitingTime[i] + proc[i].burst;
    }

    // Print the results
    printf("Process ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].id, proc[i].burst, proc[i].priority,
               waitingTime[i], turnaroundTime[i]);
    }
}

int main() {
    struct Process proc[] = {
        {1, 10, 2},
        {2, 5, 1},
        {3, 8, 3}
    };

    int n = sizeof(proc) / sizeof(proc[0]);
    priorityScheduling(proc, n);

    return 0;
}
```

Output ===

```
eduling }
Process ID      Burst Time      Priority        Waiting Time    Turnaround
Time
2               5               1               0               5
1               10              2               5               15
3               8               3               15              23
PS C:\Users\samruddhi.khilani\Desktop\VIT\OS>
```

2. 2Q] Priority Non-Preemptive CPU Scheduling Algorithm.

```
#include <stdio.h>


struct Process {
    int id;       // Process ID
    int burst;    // Burst time
    int priority; // Priority
```

```
};

void sortProcesses(struct Process proc[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            // Sort by priority (lower number = higher priority)
            if (proc[j].priority > proc[j + 1].priority) {
                struct Process temp = proc[j];
                proc[j] = proc[j + 1];
                proc[j + 1] = temp;
            }
        }
    }
}

void priorityScheduling(struct Process proc[], int n) {
    sortProcesses(proc, n);

    int waitingTime[n];
    int turnaroundTime[n];

    // Calculate waiting time
    waitingTime[0] = 0; // First process has no waiting time
    for (int i = 1; i < n; i++) {
        waitingTime[i] = waitingTime[i - 1] + proc[i - 1].burst;
    }

    // Calculate turnaround time
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = waitingTime[i] + proc[i].burst;
    }

    // Print the results
    printf("Process ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
```

```
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].id, proc[i].burst, proc[i].priority,
               waitingTime[i], turnaroundTime[i]);
    }
}

int main() {
    struct Process proc[] = {
        {1, 10, 2},
        {2, 5, 1},
        {3, 8, 3}
    };

    int n = sizeof(proc) / sizeof(proc[0]);
    priorityScheduling(proc, n);

    return 0;
}
```

Output ===

Process ID	Burst Time	Priority	Waiting Time	Turnaround Time
2	5	1	0	5
1	10	2	5	15
3	8	3	15	23

3. 1Q] SJF Preemptive CPU Scheduling Algorithm.

```
#include <stdio.h>

struct Process {
    int id;       // Process ID
    int burst;    // Burst time
    int remaining; // Remaining time
    int arrival;  // Arrival time
};

void sjfPreemptive(struct Process proc[], int n) {
    int time = 0, completed = 0;
    int waitingTime[n], turnaroundTime[n];
    int shortestIndex;
```

```
    // Initialize remaining time
    for (int i = 0; i < n; i++) {
        proc[i].remaining = proc[i].burst;
    }

    while (completed < n) {
        shortestIndex = -1;

        // Find the process with the shortest remaining time
        for (int i = 0; i < n; i++) {
            if (proc[i].remaining > 0 && proc[i].arrival <= time) {
                if (shortestIndex == -1 || proc[i].remaining < proc[shortestIndex].remaining) {
                    shortestIndex = i;
                }
            }
        }

        if (shortestIndex != -1) {
            // Execute the selected process
            proc[shortestIndex].remaining--;
            time++;

            // If process is completed
            if (proc[shortestIndex].remaining == 0) {
                completed++;
                waitingTime[shortestIndex] = time - proc[shortestIndex].burst - proc[shortestIndex].arrival;
                turnaroundTime[shortestIndex] = time - proc[shortestIndex].arrival;
            }
        } else {
            time++; // No process is ready to execute
        }
    }

    // Print the results
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].id, proc[i].burst, waitingTime[i], turnaroundTime[i]);
    }
}

int main() {
    struct Process proc[] = {
        {1, 6, 0},
        {2, 8, 1},
        {3, 7, 2}
    };

    int n = sizeof(proc) / sizeof(proc[0]);
    sjfPreemptive(proc, n);

    return 0;
}
```

Output ===

```
Process ID      Burst Time      Waiting Time    Turnaround Time
1               6               0               6
2               8               13              21
3               7               6               13
```

3.2 Q] SJF Non-Preemptive CPU Scheduling Algorithm.

```
#include <stdio.h>

struct Process {
   int id;       // Process ID
   int burst;    // Burst time
   int arrival;  // Arrival time
};

void sjfNonPreemptive(struct Process proc[], int n) {
   int waitingTime[n], turnaroundTime[n];
   int completed[n];
   int time = 0, completedCount = 0;

   // Initialize completed array
   for (int i = 0; i < n; i++) {
      completed[i] = 0;
   }

   while (completedCount < n) {
      int minIndex = -1;
      int minBurst = 9999; // A large number to find the minimum

      // Find the process with the shortest burst time that has arrived
      for (int i = 0; i < n; i++) {
         if (proc[i].arrival <= time && !completed[i] && proc[i].burst < minBurst) {
            minBurst = proc[i].burst;
            minIndex = i;
         }
      }

      if (minIndex != -1) {
         time += proc[minIndex].burst; // Execute the selected process
         completed[minIndex] = 1;
         completedCount++;

         // Calculate waiting and turnaround times
         waitingTime[minIndex] = time - proc[minIndex].arrival - proc[minIndex].burst;
         turnaroundTime[minIndex] = time - proc[minIndex].arrival;
      } else {
         time++; // No process is ready to execute
      }
   }
```

```
    // Print the results
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].id, proc[i].burst, waitingTime[i], turnaroundTime[i]);
    }
}

int main() {
    struct Process proc[] = {
        {1, 6, 0},
        {2, 8, 1},
        {3, 7, 2}
    };

    int n = sizeof(proc) / sizeof(proc[0]);
    sjfNonPreemptive(proc, n);

    return 0;
}
```

Output ==

```
Process ID      Burst Time      Waiting Time    Turnaround Time
1               6               0               6
2               8               12              20
3               7               4               11
```

3. 1Q] Round Robin CPU Scheduling Algorithm.

```
#include <stdio.h>

struct Process {
    int id;       // Process ID
    int burst;    // Burst time
    int remaining; // Remaining time
    int arrival;  // Arrival time
};

void roundRobin(struct Process proc[], int n, int quantum) {
    int waitingTime[n], turnaroundTime[n];
    int time = 0, completed = 0;

    // Initialize remaining time
    for (int i = 0; i < n; i++) {
        proc[i].remaining = proc[i].burst;
    }

    // Round Robin Scheduling
    while (completed < n) {
        int executed = 0;
```

```
        for (int i = 0; i < n; i++) {
            if (proc[i].remaining > 0 && proc[i].arrival <= time) {
                executed = 1;

                if (proc[i].remaining > quantum) {
                    // Process runs for the quantum time
                    time += quantum;
                    proc[i].remaining -= quantum;
                } else {
                    // Process finishes execution
                    time += proc[i].remaining;
                    waitingTime[i] = time - proc[i].burst - proc[i].arrival;
                    turnaroundTime[i] = time - proc[i].arrival;
                    proc[i].remaining = 0;
                    completed++;
                }
            }
        }

        if (!executed) {
            // If no process was executed, move time forward
            time++;
        }
    }

    // Print the results
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].id, proc[i].burst, waitingTime[i], turnaroundTime[i]);
    }
}

int main() {
    struct Process proc[] = {
        {1, 10, 0},
        {2, 5, 1},
        {3, 8, 2}
    };

    int n = sizeof(proc) / sizeof(proc[0]);
    int quantum = 3; // Define the time quantum
    roundRobin(proc, n, quantum);

    return 0;
}
```

Output ===

```
Process ID      Burst Time      Waiting Time    Turnaround Time
1               10              13              23
2               5               9               14
3               8               14              22
```