Lab Assignment 6

Submitted by: Isha Gupta
Roll No: 102303007
Sub Group: 2C11

Question 1. Write a program using C/C++/Java to simulate the priority scheduling (pre-emptive as well as non-pre-emptive approach) and RR, CPU scheduling algorithms. The scenario is: user may input n processes with respective CPU burst time and arrival time (also take the priority number in case of priority scheduling). System will ask the user to select the type of algorithm from the list mentioned above. System should display the waiting time for each process, average waiting time for the whole system, and final execution sequence.

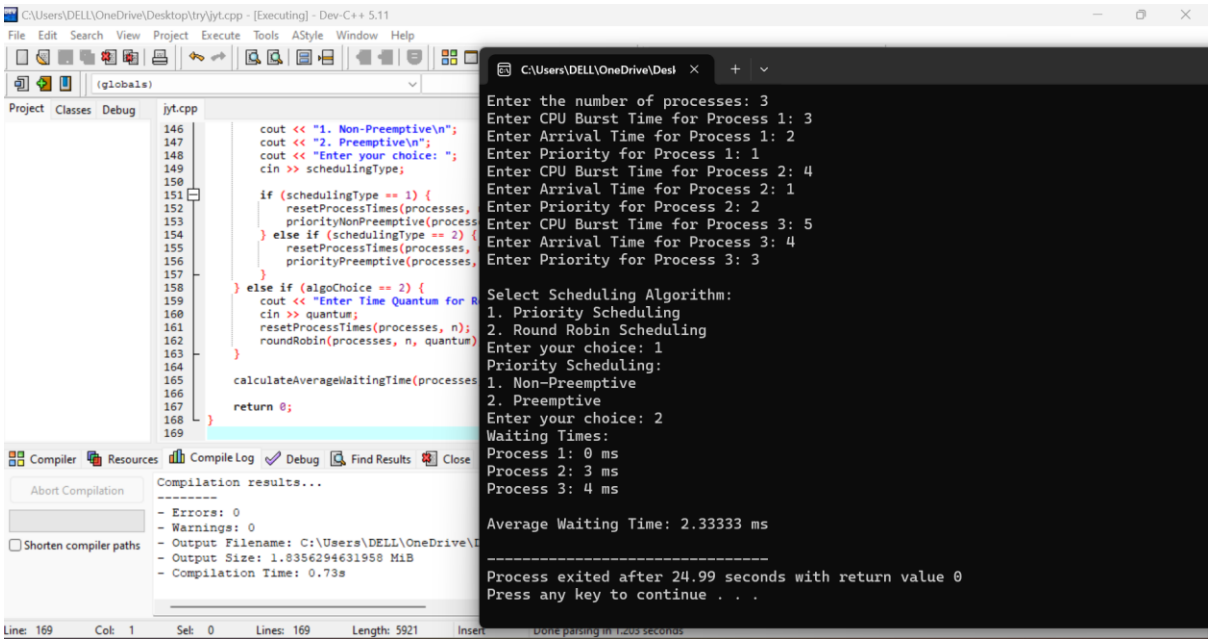
Ans.

```
#include <iostream>
using namespace std;

struct Process {
   int pid; // Process ID
   int burstTime; // CPU burst time
   int arrivalTime; // Arrival time
   int priority; // Priority for priority scheduling
   int waitingTime; // Waiting time
   int turnaroundTime; // Turnaround time
   int remainingTime; // Remaining burst time (for pre-emptive)
};

// Function to calculate waiting time for non-preemptive Priority Scheduling
void priorityNonPreemptive(Process processes[], int n) {
   int completed = 0, currentTime = 0, minPriority, minIndex;
   bool selected;

   while (completed != n) {
      minPriority = 9999;
      selected = false;

      // Find the process with the highest priority (smallest priority number)
      for (int i = 0; i < n; i++) {
         if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0) {
            if (processes[i].priority < minPriority) {
               minPriority = processes[i].priority;
               minIndex = i;
               selected = true;
            }
         }
      }
```

```
        if (selected) {
            currentTime += processes[minIndex].remainingTime;
            processes[minIndex].waitingTime = currentTime - processes[minIndex].arrivalTime -
processes[minIndex].burstTime;
            processes[minIndex].remainingTime = 0;
            completed++;
        } else {
            currentTime++;
        }
    }
}

// Function to calculate waiting time for preemptive Priority Scheduling
void priorityPreemptive(Process processes[], int n) {
    int completed = 0, currentTime = 0, minPriority, minIndex;
    bool selected;

    while (completed != n) {
        minPriority = 9999;
        selected = false;

        // Find the process with the highest priority (smallest priority number)
        for (int i = 0; i < n; i++) {
            if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0) {
                if (processes[i].priority < minPriority) {
                    minPriority = processes[i].priority;
                    minIndex = i;
                    selected = true;
                }
            }
        }

        if (selected) {
            processes[minIndex].remainingTime--;
            currentTime++;

            if (processes[minIndex].remainingTime == 0) {
                processes[minIndex].waitingTime = currentTime - processes[minIndex].arrivalTime -
processes[minIndex].burstTime;
                completed++;
            }
        } else {
            currentTime++;
        }
    }
}

// Function to calculate waiting time for Round Robin Scheduling
```

```
void roundRobin(Process processes[], int n, int quantum) {
   int currentTime = 0, completed = 0;
   int i = 0;
   while (completed != n) {
      if (processes[i].remainingTime > 0 && processes[i].arrivalTime <= currentTime) {
         if (processes[i].remainingTime > quantum) {
            processes[i].remainingTime -= quantum;
            currentTime += quantum;
         } else {
            currentTime += processes[i].remainingTime;
            processes[i].remainingTime = 0;
            processes[i].waitingTime = currentTime - processes[i].arrivalTime - processes[i].burstTime;
            completed++;
         }
      }
      i = (i + 1) % n;
      if (i == 0 && completed < n && processes[i].remainingTime == 0)
         currentTime++;
   }
}

// Function to calculate average waiting time
void calculateAverageWaitingTime(Process processes[], int n) {
   int totalWaitingTime = 0;
   cout << "Waiting Times: \n";
   for (int i = 0; i < n; i++) {
      totalWaitingTime += processes[i].waitingTime;
      cout << "Process " << processes[i].pid << ": " << processes[i].waitingTime << " ms\n";
   }
   cout << "\nAverage Waiting Time: " << (float)totalWaitingTime / n << " ms\n";
}

// Function to reset remaining time and waiting time for each process
void resetProcessTimes(Process processes[], int n) {
   for (int i = 0; i < n; i++) {
      processes[i].remainingTime = processes[i].burstTime;
      processes[i].waitingTime = 0;
   }
}

int main() {
   int n, quantum, algoChoice, schedulingType;
   cout << "Enter the number of processes: ";
   cin >> n;

   Process processes[n];

   for (int i = 0; i < n; i++) {
```

```
        processes[i].pid = i + 1;
        cout << "Enter CPU Burst Time for Process " << processes[i].pid << ": ";
        cin >> processes[i].burstTime;
        cout << "Enter Arrival Time for Process " << processes[i].pid << ": ";
        cin >> processes[i].arrivalTime;
        cout << "Enter Priority for Process " << processes[i].pid << ": ";
        cin >> processes[i].priority;
        processes[i].remainingTime = processes[i].burstTime;
    }

    cout << "\nSelect Scheduling Algorithm:\n";
    cout << "1. Priority Scheduling\n";
    cout << "2. Round Robin Scheduling\n";
    cout << "Enter your choice: ";
    cin >> algoChoice;

    if (algoChoice == 1) {
        cout << "Priority Scheduling:\n";
        cout << "1. Non-Preemptive\n";
        cout << "2. Preemptive\n";
        cout << "Enter your choice: ";
        cin >> schedulingType;

        if (schedulingType == 1) {
            resetProcessTimes(processes, n);
            priorityNonPreemptive(processes, n);
        } else if (schedulingType == 2) {
            resetProcessTimes(processes, n);
            priorityPreemptive(processes, n);
        }
    } else if (algoChoice == 2) {
        cout << "Enter Time Quantum for Round Robin: ";
        cin >> quantum;
        resetProcessTimes(processes, n);
        roundRobin(processes, n, quantum);
    }

    calculateAverageWaitingTime(processes, n);

    return 0;
}
```


C:\Users\DELL\OneDrive\Desktop\try\jyt.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug
jyt.cpp
146 cout << "1. Non-Preemptive\n";
147 cout << "2. Preemptive\n";
148 cout << "Enter your choice: ";
149 cin >> schedulingType;
150
151 if (schedulingType == 1) {
152 resetProcessTimes(processes,
153 priorityNonPreemptive(process
154 } else if (schedulingType == 2) {
155 resetProcessTimes(processes,
156 priorityPreemptive(processes,
157 }
158 } else if (algoChoice == 2) {
159 cout << "Enter Time Quantum for R
160 cin >> quantum;
161 resetProcessTimes(processes, n);
162 roundRobin(processes, n, quantum)
163 }
164
165 calculateAverageWaitingTime(processes
166
167 return 0;
168 }
169
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\DELL\OneDrive\D
- Output Size: 1.8356294631958 MiB
- Compilation Time: 0.73s
Line: 169 Col: 1 Sel: 0 Lines: 169 Length: 5921 Insert
C:\Users\DELL\OneDrive\Desk
Enter the number of processes: 3
Enter CPU Burst Time for Process 1: 3
Enter Arrival Time for Process 1: 2
Enter Priority for Process 1: 1
Enter CPU Burst Time for Process 2: 4
Enter Arrival Time for Process 2: 1
Enter Priority for Process 2: 2
Enter CPU Burst Time for Process 3: 5
Enter Arrival Time for Process 3: 4
Enter Priority for Process 3: 3
Select Scheduling Algorithm:
1. Priority Scheduling
2. Round Robin Scheduling
Enter your choice: 1
Priority Scheduling:
1. Non-Preemptive
2. Preemptive
Enter your choice: 2
Waiting Times:
Process 1: 0 ms
Process 2: 3 ms
Process 3: 4 ms
Average Waiting Time: 2.33333 ms

Process exited after 24.99 seconds with return value 0
Press any key to continue . . .