OS

lab#9

Name: Anas-Altaf

Roll.No: 22F-3639

Code:

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>
#include <string>
#include <iomanip>

using namespace std;

struct Process
{
   string name;
   int arrivalTime;
   int burstTime;
   int remainingTime;
   int priority;
   int startTime;
   int endTime;
   int waitingTime;
   int turnaroundTime;

   bool operator<(const Process &other) const
   {
       if (arrivalTime == other.arrivalTime)
           return burstTime > other.burstTime;
       return arrivalTime > other.arrivalTime;
   }
};
```

```
void displayGanttChart(const vector<string> &ganttChart)
{
   cout << "\nGantt Chart:\n";
   for (const auto &name : ganttChart)
   {
       cout << "| " << name << " ";
   }
   cout << "|\n";
}

int main()
{
   int numProcesses, timeQuantum1 = 4, timeQuantum2 = 8;
   cout << "Enter the number of processes: ";
   cin >> numProcesses;

   vector<Process> processes(numProcesses);
   vector<string> ganttChart;

   for (int i = 0; i < numProcesses; i++)
   {
       cout << "Enter process name (e.g., P1): ";
       cin >> processes[i].name;
       cout << "Enter arrival time of " << processes[i].name << ": ";
       cin >> processes[i].arrivalTime;
       cout << "Enter burst time of " << processes[i].name << ": ";
       cin >> processes[i].burstTime;
       processes[i].remainingTime = processes[i].burstTime;
       cout << "Enter priority of " << processes[i].name << " (lower value
means higher priority): ";
       cin >> processes[i].priority;
       processes[i].waitingTime = 0;
       processes[i].turnaroundTime = 0;
   }

   sort(processes.begin(), processes.end(), [](const Process &a, const
Process &b)
        { return a.arrivalTime < b.arrivalTime; });

   queue<Process *> q1, q2, q3;
```

```
    int currentTime = 0;
    int completedProcesses = 0;

    while (completedProcesses < numProcesses)
    {

        for (auto &process : processes)
        {
            if (process.arrivalTime <= currentTime && process.remainingTime
> 0)
            {
                if (process.priority == 1)
                    q1.push(&process);
                else if (process.priority == 2)
                    q2.push(&process);
                else
                    q3.push(&process);
            }
        }

        if (!q1.empty())
        {
            Process *p = q1.front();
            q1.pop();
            ganttChart.push_back(p->name);

            if (p->remainingTime > timeQuantum1)
            {
                currentTime += timeQuantum1;
                p->remainingTime -= timeQuantum1;

                q2.push(p);
            }
            else
            {
                currentTime += p->remainingTime;
                p->endTime = currentTime;
                p->turnaroundTime = p->endTime - p->arrivalTime;
                p->waitingTime = p->turnaroundTime - p->burstTime;
                p->remainingTime = 0;
```

```
            completedProcesses++;
        }
    }

    else if (!q2.empty())
    {
        Process *p = q2.front();
        q2.pop();
        ganttChart.push_back(p->name);

        if (p->remainingTime > timeQuantum2)
        {
            currentTime += timeQuantum2;
            p->remainingTime -= timeQuantum2;

            q3.push(p);
        }
        else
        {
            currentTime += p->remainingTime;
            p->endTime = currentTime;
            p->turnaroundTime = p->endTime - p->arrivalTime;
            p->waitingTime = p->turnaroundTime - p->burstTime;
            p->remainingTime = 0;
            completedProcesses++;
        }
    }

    else if (!q3.empty())
    {
        Process *p = q3.front();
        q3.pop();
        ganttChart.push_back(p->name);

        currentTime += p->remainingTime;
        p->endTime = currentTime;
        p->turnaroundTime = p->endTime - p->arrivalTime;
        p->waitingTime = p->turnaroundTime - p->burstTime;
        p->remainingTime = 0;
        completedProcesses++;
```

```
        }
        else
        {

            currentTime++;
        }
    }

    displayGanttChart(ganttChart);

    cout << "\nProcess\tArrival\tBurst\tPriority\tWaiting\tTurnaround\n";
    for (const auto &p : processes)
    {
        cout << p.name << "\t" << p.arrivalTime << "\t" << p.burstTime <<
"\t" << p.priority
             << "\t\t" << max(0, p.waitingTime) << "\t" << max(0,
p.turnaroundTime) << "\n";
    }

    return 0;
}
```

Output:

```
Enter arrival time of p1: Enter burst time of p1: Enter priority of p1 (lower value means higher priority): Enter process name (e.g., P1)
: Enter arrival time of 4: 2
Enter burst time of 4: 3
Enter priority of 4 (lower value means higher priority): 4
Enter process name (e.g., P1): p3
Enter arrival time of p3: 2 5 7
Enter burst time of p3: Enter priority of p3 (lower value means higher priority):
Gantt Chart:
| p1 | 4 | p3 |

Process Arrival Burst   Priority        Waiting Turnaround
p1      0       2       3               0       2
4       2       3       4               0       3
p3      2       5       7               3       8
[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-bv3yrs3s.sxr" 1>"/tmp/Mic
rosoft-MIEngine-Out-bsnwfrjd.p5x"
```