ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC VÀ KỸ THUẬT THÔNG TIN

HỆ ĐIỀU HÀNH
IT007.O21.LT

BÁO CÁO
BÁO CÁO THỰC HÀNH LAB 5

Sinh viên thực hiện:
Trần Duy Tân - 22550020

Giảng viên:
ThS. Đỗ Trí Nhựt

Thành phố Hồ Chí Minh, năm 2024

Viết chương trình mô phỏng giải thuật Round Robin với các yêu cầu sau:

- ❖ Nhập số lượng process
- ❖ Nhập quantum time
- ❖ Nhập process name, burst time
- ❖ In ra Gantt chart, với các thông số Process name, start process time, stop process time.
- ❖ In ra average waiting time, average turnaround time

Test Case

Giản đồ Gantt (quantum time = 4)

Process	Arrival Time	Burst Time
P1	0	12
P2	2	7
P3	5	8
P4	9	3
P5	12	6

Kết quả:

```
Enter quantum time: 4
Enter the number of processes: 5
==========
Enter process name: P1
Enter arrival time P1: 0
Enter burst time P1: 12
==========
Enter process name: P2
Enter arrival time P2: 2
Enter burst time P2: 7
==========
Enter process name: P3
Enter arrival time P3: 5
Enter burst time P3: 8
==========
Enter process name: P4
Enter arrival time P4: 9
==========
Enter process name: P5
Enter arrival time P5: 12
Enter burst time P5: 6
==========
Result:
Process Response Time   Waiting Time    Turnaround Time
P1      0               0               12
P2      10              10              17
P3      14              14              22
P4      18              18              21
P5      18              18              24

Gantt chart:
|== P1 ==|== P1 ==|== P1 ==|== P2 ==|== P2 ==|== P3 ==|== P3 ==|== P4 ==|== P5 ==|== P5 ==|
0     4     8     12     16     19     23     27     30     34     36
```

Code demo

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

struct Process {
    std::string name;
    int arrival_time;
    int burst_time;
    int response_time = -1;
    int waiting_time = 0;
    int turnaround_time = 0;
};

struct Point {
    std::string name;
    int time;
};

bool compareObjects(const Process &obj1, const Process &obj2) {
    return obj1.arrival_time < obj2.arrival_time;
}

void checkNewProcess(std::vector<Process> &processes, std::vector<int> &queue,
int completed_processes, int current_time, int checkingTime) {
    for (int i = completed_processes; i < processes.size(); ++i) {
        if (processes[i].arrival_time <= current_time &&
processes[i].arrival_time > (current_time - checkingTime)) {
            queue.push_back(i);
        }
    }
}

void roundRobinScheduling(std::vector<Process> &processes, int quantum,
std::vector<Point> &chart) {
    std::vector<int> remaining_time(processes.size());
    std::vector<int> completion_time(processes.size(), 0);
    int current_time = 0;
    int completed_processes = 0;

    for (int i = 0; i < processes.size(); ++i) {
        remaining_time[i] = processes[i].burst_time;
    }

    std::vector<int> queue;
    queue.push_back(0);
    current_time = processes[0].arrival_time;
```

```
    chart.push_back({processes[0].name, processes[0].arrival_time});
    processes[0].response_time = 0;

    while (!queue.empty()) {
        int idx = queue.front();
        queue.erase(queue.begin());

        if (processes[idx].response_time == -1) {
            processes[idx].response_time = current_time -
processes[idx].arrival_time;
        }

        if (remaining_time[idx] <= quantum) {
            current_time += remaining_time[idx];
            chart.push_back({processes[idx].name, current_time});
            remaining_time[idx] = 0;
            completion_time[idx] = current_time;
            completed_processes++;
        } else {
            current_time += quantum;
            chart.push_back({processes[idx].name, current_time});
            remaining_time[idx] -= quantum;
        }

        checkNewProcess(processes, queue, completed_processes, current_time,
quantum);

        if (remaining_time[idx] > 0) {
            queue.push_back(idx);
        }

        std::sort(queue.begin(), queue.end(), [&processes](int a, int b) {
            return processes[a].arrival_time < processes[b].arrival_time;
        });
    }

    for (int i = 0; i < processes.size(); ++i) {
        processes[i].turnaround_time = completion_time[i] -
processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
    }
}

int main() {
    int num_processes, quantum;
    std::cout << "Enter quantum time: ";
    std::cin >> quantum;
```

```
    std::cout << "Enter the number of processes: ";
    std::cin >> num_processes;
    std::cout << "==========\n";

    std::vector<Process> processes(num_processes);
    std::vector<Point> chart;

    for (int i = 0; i < num_processes; ++i) {
        std::cout << "Enter process name: ";
        std::cin >> processes[i].name;
        std::cout << "Enter arrival time " << processes[i].name << ": ";
        std::cin >> processes[i].arrival_time;
        std::cout << "Enter burst time " << processes[i].name << ": ";
        std::cin >> processes[i].burst_time;
        std::cout << "==========\n";
    }

    std::sort(processes.begin(), processes.end(), compareObjects);
    roundRobinScheduling(processes, quantum, chart);

    double total_waiting_time = 0;
    double total_turnaround_time = 0;

    std::cout << "Result:\n";
    std::cout << "Process\tResponse Time\tWaiting Time\tTurnaround Time\n";
    for (const auto &process : processes) {
        std::cout << process.name << "\t" << process.response_time << "\t\t"
<< process.waiting_time << "\t\t" << process.turnaround_time << "\n";
        total_waiting_time += process.waiting_time;
        total_turnaround_time += process.turnaround_time;
    }

    double average_waiting_time = total_waiting_time / num_processes;
    double average_turnaround_time = total_turnaround_time / num_processes;

    std::cout << "\nGantt chart:\n";
    std::cout << "|";
    for (int i = 1; i < chart.size(); ++i) {
        std::cout << "== " << chart[i].name << " ==|";
    }
    std::cout << "\n";
    for (const auto &item : chart) {
        std::cout << item.time << "     ";
    }
    std::cout << "\n\nAverage Waiting Time: " << average_waiting_time << "\n";
    std::cout << "Average Turnaround Time: " << average_turnaround_time <<
"\n";
```

```
    return 0;
}
```