**LAB # 11 & 12**

****

**Spring 2022**

**Operating System**

**Submitted By: Maaz Habib, Alishba Orakzai, Nida Asmat Burki**

**Registration No: 20PWCSE1952, 20pwcse1953, 20pwcse1955**

**Class Section:**

**C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Submitted to:

**Engr Mian Ibad**

July 09, 2022

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab11-Task1:**

**Code:**

#include <iostream>

#include <algorithm>

#include <iomanip>

using namespace std;

struct process {

int pid;

int arrival\_time;

int burst\_time;

int start\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

bool compareArrival(process p1, process p2)

{

return p1.arrival\_time < p2.arrival\_time;

}

bool compareID(process p1, process p2)

{

return p1.pid < p2.pid;

}

int main() {

int n;

struct process p[100];

float avg\_turnaround\_time;

float avg\_waiting\_time;

float avg\_response\_time;

float cpu\_utilisation;

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

int total\_idle\_time = 0;

float throughput;

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";

cin>>n;

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

cout<<"Enter arrival time of process "<<i+1<<": ";

cin>>p[i].arrival\_time;

cout<<"Enter burst time of process "<<i+1<<": ";

cin>>p[i].burst\_time;

p[i].pid = i+1;

cout<<endl;

}

sort(p,p+n,compareArrival);

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

p[i].start\_time = (i == 0)?p[i].arrival\_time:max(p[i-1].completion\_time,p[i].arrival\_time);

p[i].completion\_time = p[i].start\_time + p[i].burst\_time;

p[i].turnaround\_time = p[i].completion\_time - p[i].arrival\_time;

p[i].waiting\_time = p[i].turnaround\_time - p[i].burst\_time;

p[i].response\_time = p[i].start\_time - p[i].arrival\_time;

total\_turnaround\_time += p[i].turnaround\_time;

total\_waiting\_time += p[i].waiting\_time;

total\_response\_time += p[i].response\_time;

total\_idle\_time += (i == 0)?(p[i].arrival\_time):(p[i].start\_time - p[i-1].completion\_time);

}

avg\_turnaround\_time = (float) total\_turnaround\_time / n;

avg\_waiting\_time = (float) total\_waiting\_time / n;

avg\_response\_time = (float) total\_response\_time / n;

cpu\_utilisation = ((p[n-1].completion\_time - total\_idle\_time) / (float) p[n-1].completion\_time)\*100;

throughput = float(n) / (p[n-1].completion\_time - p[0].arrival\_time);

sort(p,p+n,compareID);

cout<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

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

cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

}

cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

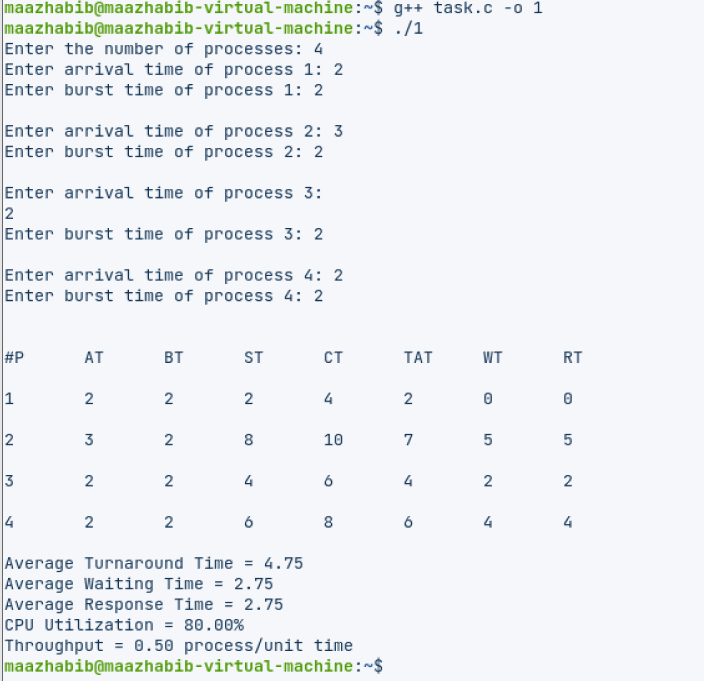
cout<<"Average Response Time = "<<avg\_response\_time<<endl;

cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;

}

**Output:**



**Lab11-Task2:**

**Code:**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

int pid;

int arrival\_time;

int burst\_time;

int start\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

int main() {

int n;

struct process p[100];

float avg\_turnaround\_time;

float avg\_waiting\_time;

float avg\_response\_time;

float cpu\_utilisation;

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

int total\_idle\_time = 0;

float throughput;

int is\_completed[100];

memset(is\_completed,0,sizeof(is\_completed));

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";

cin>>n;

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

cout<<"Enter arrival time of process "<<i+1<<": ";

cin>>p[i].arrival\_time;

cout<<"Enter burst time of process "<<i+1<<": ";

cin>>p[i].burst\_time;

p[i].pid = i+1;

cout<<endl;

}

int current\_time = 0;

int completed = 0;

int prev = 0;

while(completed != n) {

int idx = -1;

int mn = 10000000;

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

if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

if(p[i].burst\_time < mn) {

mn = p[i].burst\_time;

idx = i;

}

if(p[i].burst\_time == mn) {

if(p[i].arrival\_time < p[idx].arrival\_time) {

mn = p[i].burst\_time;

idx = i;

}

}

}

}

if(idx != -1) {

p[idx].start\_time = current\_time;

p[idx].completion\_time = p[idx].start\_time + p[idx].burst\_time;

p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

total\_turnaround\_time += p[idx].turnaround\_time;

total\_waiting\_time += p[idx].waiting\_time;

total\_response\_time += p[idx].response\_time;

total\_idle\_time += p[idx].start\_time - prev;

is\_completed[idx] = 1;

completed++;

current\_time = p[idx].completion\_time;

prev = current\_time;

}

else {

current\_time++;

}

}

int min\_arrival\_time = 10000000;

int max\_completion\_time = -1;

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

min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

}

avg\_turnaround\_time = (float) total\_turnaround\_time / n;

avg\_waiting\_time = (float) total\_waiting\_time / n;

avg\_response\_time = (float) total\_response\_time / n;

cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

throughput = float(n) / (max\_completion\_time - min\_arrival\_time);

cout<<endl<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

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

cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

}

cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

cout<<"Average Response Time = "<<avg\_response\_time<<endl;

cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;

}

/\*

AT - Arrival Time of the process

BT - Burst time of the process

ST - Start time of the process

CT - Completion time of the process

TAT - Turnaround time of the process

WT - Waiting time of the process

RT - Response time of the process

Formulas used:

TAT = CT - AT

WT = TAT - BT

RT = ST - AT

\*/

**OutPut:**

****

**Lab12-Task1:**

**Code:**

#include<iostream>

#include<queue>

#include<vector>

#include<algorithm>

#define TQ 2

#define NB\_PROCESS 7

using namespace std;

class Process{

public:

int process\_id;

int arrival\_time;

int burst\_time;

int remaining\_time;

int completion\_time = 0;

int turn\_around\_time = -1;

int waiting\_time = -1;

Process(int p\_id, int a\_time, int b\_time){

this->process\_id = p\_id;

this->arrival\_time = a\_time;

this->burst\_time = b\_time;

this->remaining\_time = this->burst\_time;

}

bool operator!=(Process& p){

if (this->process\_id == p.process\_id) return false;

return true;

}

};

int main(){

int process\_id[] = {0, 1, 2, 3, 4, 5, 6};

int arrival\_time[] = {9, 1, 4, 5, 2, 30, 29};

int burst\_time[] = {10, 2, 1, 5, 7, 3, 6};

vector<Process> process\_list;

queue<Process\*> ready\_queue;

vector<Process> gantt;

Process idle(-1,0,0);

for(int i=0; i<NB\_PROCESS; i++)

process\_list.push\_back(Process(process\_id[i], arrival\_time[i], burst\_time[i]));

// Bubble sort on arrival\_time:

for(int i=0; i<NB\_PROCESS; i++)

for (int j=1; j<NB\_PROCESS-i; j++)

if (process\_list.at(j).arrival\_time < process\_list.at(j-1).arrival\_time){

Process temp = process\_list.at(j);

process\_list.at(j) = process\_list.at(j-1);

process\_list.at(j-1)= temp;

}

if(process\_list.at(0).arrival\_time > 0)

idle.completion\_time = process\_list.at(0).arrival\_time;

gantt.push\_back(idle);

ready\_queue.push(&process\_list.at(0));

while( !ready\_queue.empty() ){

Process\* current = ready\_queue.front();

ready\_queue.pop();

if(current->remaining\_time <= TQ){

current->completion\_time = gantt.back().completion\_time + current->remaining\_time;

current->remaining\_time = 0;

}

else{

current->completion\_time = gantt.back().completion\_time + TQ;

current->remaining\_time = current->remaining\_time - TQ;

}

for(Process& p: process\_list){

if(p.arrival\_time > gantt.back().completion\_time && p.arrival\_time <= current->completion\_time ){

ready\_queue.push(&p);

}

}

if(current->remaining\_time != 0)

ready\_queue.push(current);

gantt.push\_back(\*current);

if (ready\_queue.empty() )

for (Process& p: process\_list)

if(p.remaining\_time != 0 ){

idle.arrival\_time = gantt.back().completion\_time;

idle.completion\_time = p.arrival\_time;

gantt.push\_back(idle);

ready\_queue.push(&p);

break;

}

}

cout << "CPU Scheduling:" << endl << "Round Robin Scheduling:";

printf("\n\n %15s | %15s | %15s | %15s | %15s | %15s |\n\n", "Process Id",

"Arrival Time",

"Burst Time",

"Completion Time",

"Turn Around T.",

"Waiting Time");

// Bubble sort on process\_id:

// for(int i=0; i<NB\_PROCESS; i++)

// for (int j=1; j<NB\_PROCESS-i; j++)

// if (process\_list.at(j).process\_id < process\_list.at(j-1).process\_id){

// Process temp = process\_list.at(j);

// process\_list.at(j) = process\_list.at(j-1);

// process\_list.at(j-1) = temp;

// }

reverse( gantt.begin(), gantt.end() );

for(Process& p: process\_list){

p.turn\_around\_time = p.completion\_time - p.arrival\_time;

p.waiting\_time = p.turn\_around\_time - p.burst\_time;

printf(" %14d | %14d | %14d | %14d | %14d | %14d |\n", p.process\_id,

p.arrival\_time,

p.burst\_time,

p.completion\_time,

p.turn\_around\_time,

p.waiting\_time);

}

reverse( gantt.begin(), gantt.end() );

// comment this out to print all blocks separately

for(int i=gantt.size()-1; i>0; i--){

if(gantt.at(i).process\_id == gantt.at(i-1).process\_id){

gantt.erase(gantt.begin()+i-1);

}

}

// till here

cout << "\n\nGantt Chart:\n ";

for (Process& p: gantt)

if (p.process\_id == -1){

if (p.completion\_time != 0)

printf(" idle |");

}

else

printf(" P%d |", p.process\_id);

cout << endl << 0;

for (Process& p: gantt)

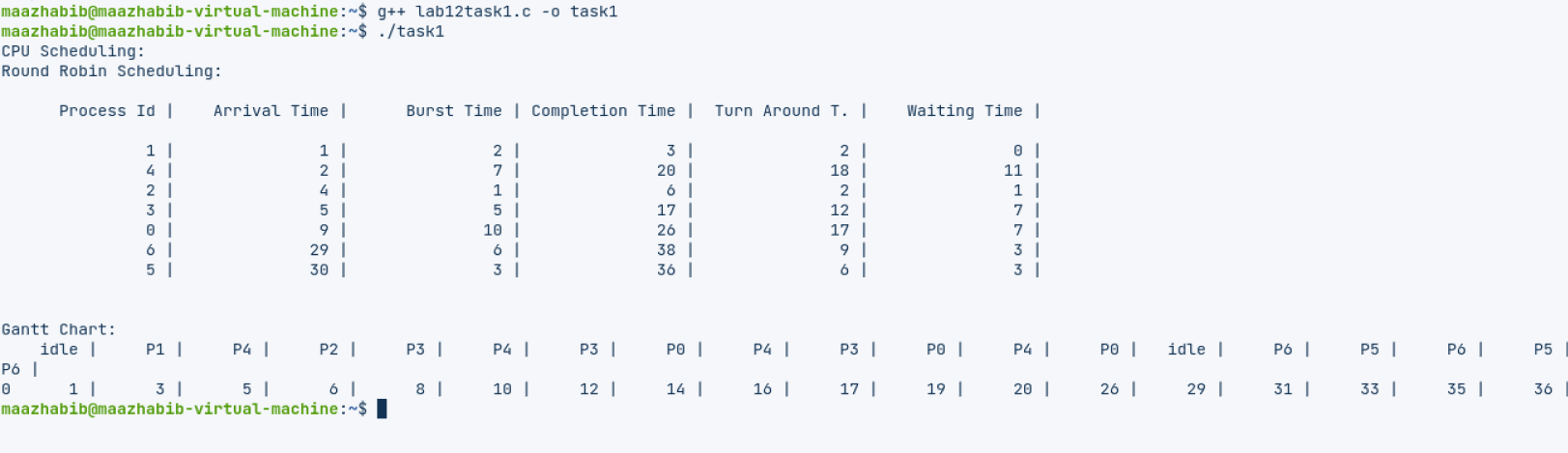
if (p.completion\_time)

printf(" %2d |", p.completion\_time);

return 0;

}

**OutPut:**



**Lab12-Task2:**

**Code:**



**OutPut:**

