Name: Phạm Đức Thể

ID: 19522253

Class: IT007.M14.2

OPERATING SYSTEM
LAB 4 REPORT

SUMMARY

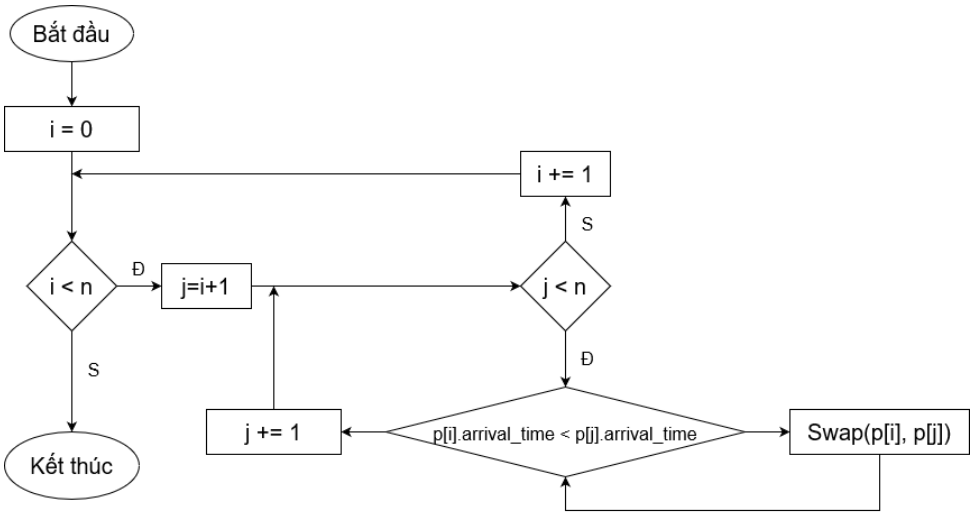
Self-scrores: 10/10

Note: Export file to **PDF and name the file by following format:*
LAB X – <Student ID>.pdf

Section 4.5

1. Task name 1: Viết chương trình mô phỏng giải thuật SJF.

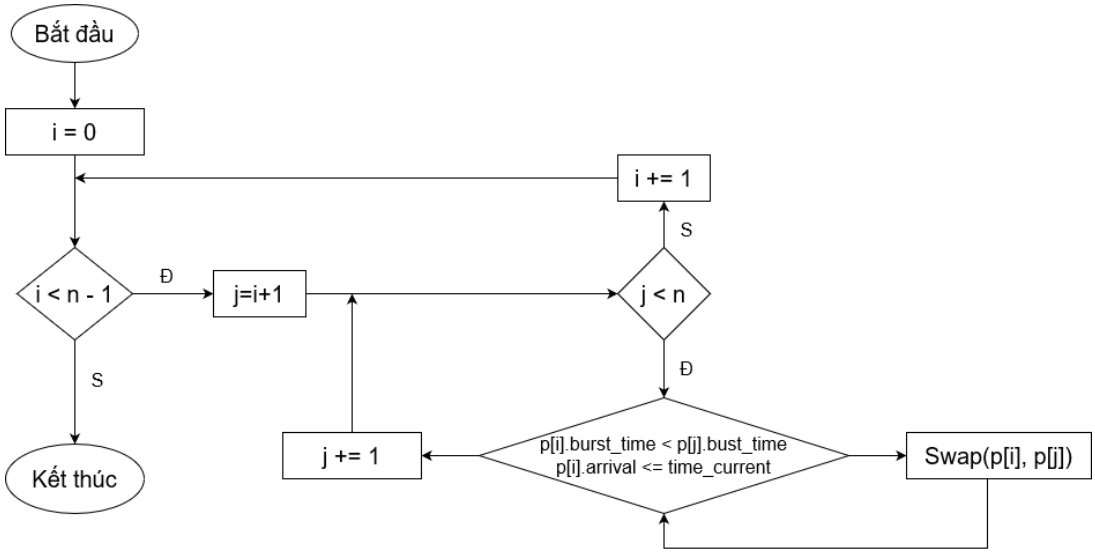
1.1. Hàm sort các tiến trình theo arrival time



Hình 1: Lưu đồ hàm sort các tiến trình dựa vào arrival_time

- **Giải thích:** Chúng ta sẽ sử dụng thuật toán nổi bọt để lọc quá hết các cặp phần tử và sắp xếp lại theo thứ tự có arrival_time giảm dần.

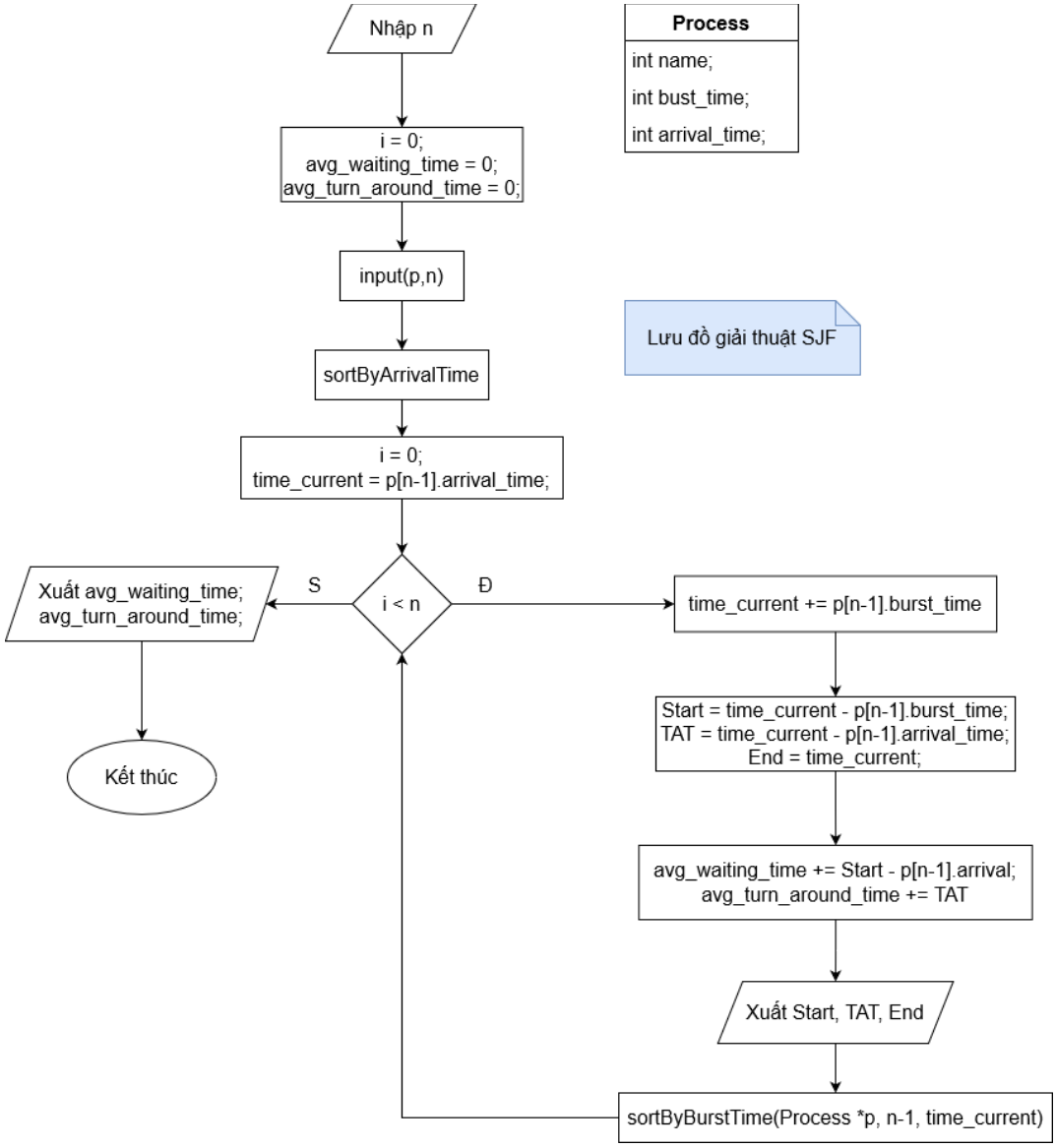
1.2. Hàm sort các tiến trình theo burst time



Hình 2: Lưu đồ hàm sort các tiến trình dựa vào burst time

- **Giải thích:** Tương tự chúng ta sẽ sử dụng thuật toán nổi bọt để lọc quá hết các cặp phần ử và sắp xếp lại các tiến trình chưa xử lý theo thứ tự có burst_time tăng dần. Và tá xét điều kiện là arrival_time phải bé hơn hoặc bằng thời gian hiện tại đang thực thi.

1.3. Lưu đồ giải thuật SJF



Hình 3: Lưu đồ giải thuật SJF

– **Giải thích:**

- Đầu tiên ta sẽ tạo ra một struct tên process với 3 thông tin cơ bản như trên. Sau đó chúng ta khai báo thêm 2 biến toàn cục là biến tổng thời gian đợi và thời gian thực hiện trong hệ thống.
- Tiến hành nhập n là số process, Sau đó dùng hàm Input để nhập các thông tin của các process.
- Sắp xếp lại các tiến trình bằng hàm SortByArrivalTime. Sau đó khai báo thêm biến time_current = thời gian vào của tiến trình có arrival_time bé nhất.

- Cho các tiến trình vào vòng lặp lấy ra phần tử ngoài cùng lúc này tiến trình đầu tiên được thực thi, time_current lúc này đã được cộng thêm busrt_time của tiến trình đó lúc này time_current là thời gian kết thúc của tiến trình trong vòng lặp.
- Tiến hành tính toán các thời gian Star, TAT, End.
- ắp xếp lại các tiến trình còn lại dựa vào hàm sortByBurstTime và lặp lại đối với các tiến trình còn lại.

1.4. Code của giải thuật

```
1  /*################
2  # University of Information Technology
3  # IT007 Operating System
4  # Pham Duc The, 19522253
5  # File: sjf.cpp
6  ################*/
7
8
9  #include <stdio.h>
10 #include <iostream>
11 #include <queue>
12
13 using namespace std;
14
15 struct Process {
16         int name;
17         int burst_time;
18         int arrival_time;
19 };
20
21 static double avg_turn_around_time = 0;
22 static double avg_waiting_time = 0;
23
24 void swap(Process &p1, Process &p2){
25         Process tmp;
26         tmp = p1;
27         p1 = p2;
28         p2 = tmp;
29 }
30
31 void sortByArrivalTime(Process *p, int n){
32         for(int i = 0; i < n; i++){
33                 for(int j = i+1; j < n; j++){
34                         if(p[i].arrival_time < p[j].arrival_time){
35                                 swap(p[i], p[j]);
36                         }
37                 }
38         }
39 }
40
41 void Input(Process *p, int n){
42         for(int i = 0; i < n; i++){
43                 cout << "----------------\n";
44                 cout << "Nhap ID process:"; cin >> p[i].name;
45                 cout << "Nhap Arrival Time:"; cin >> p[i].arrival_time;
46                 cout << "Nhap Burst Time:"; cin >> p[i].burst_time;
47         }
48 }
49
```

Hình 4: Code từ dòng 1 - 49

```
49
50 void sortByBurstTime(Process *p, int n, int time_current){
51         for(int i = 0; i < n-1; i++){
52                 for(int j = i+1; j < n; j++){
53                         if(p[i].burst_time < p[j].burst_time && p[i].arrival_time <=
   time_current){
54                                 swap(p[i], p[j]);
55                         }
56                 }
57         }
58 }
59
60 void SelectionFunction(Process *p, int n){
61         int time_current;
62         // Sort theo arrival_time
63         sortByArrivalTime(p, n);
64         // Ham lua chon quyet dinh xem process nao vao queue truoc;
65         time_current = p[n-1].arrival_time;
66         for(int i = 0; i < n; n--){
67                 time_current += p[n-1].burst_time;
68                 avg_waiting_time += time_current - p[n-1].arrival_time -
   p[n-1].burst_time;
69                 avg_turn_around_time += (time_current - p[n-1].arrival_time);
70                 cout << p[n-1].name << "\t\t" << p[n-1].arrival_time << "\t\t" <<
   p[n-1].burst_time << "\t\t" << time_current - p[n-1].burst_time << "\t\t" << time_current
   - p[n-1].arrival_time << "\t\t" << (time_current) << endl;
71                 sortByBurstTime(p, n-1, time_current);
72         }
73 }
74
75 int main(){
76         Process *p = new Process[100];
77         queue<Process> pQueue;
78         int n;
79         cout << "nhap so luong process: "; cin >> n;
80         Input(p,n);
81         cout << "PName\t\tArrtime\t\tBurtime\t\tStart\t\tTAT\t\tFinish\n";
82         SelectionFunction(p,n);
83         cout <<"Thoi gian dap ung trung binh: " <<  avg_waiting_time/n << endl;
84         cout <<"Thoi gian hoan thanh trung binh: " <<  avg_turn_around_time/n << endl;
85         return 0;
86 }
87
88
```

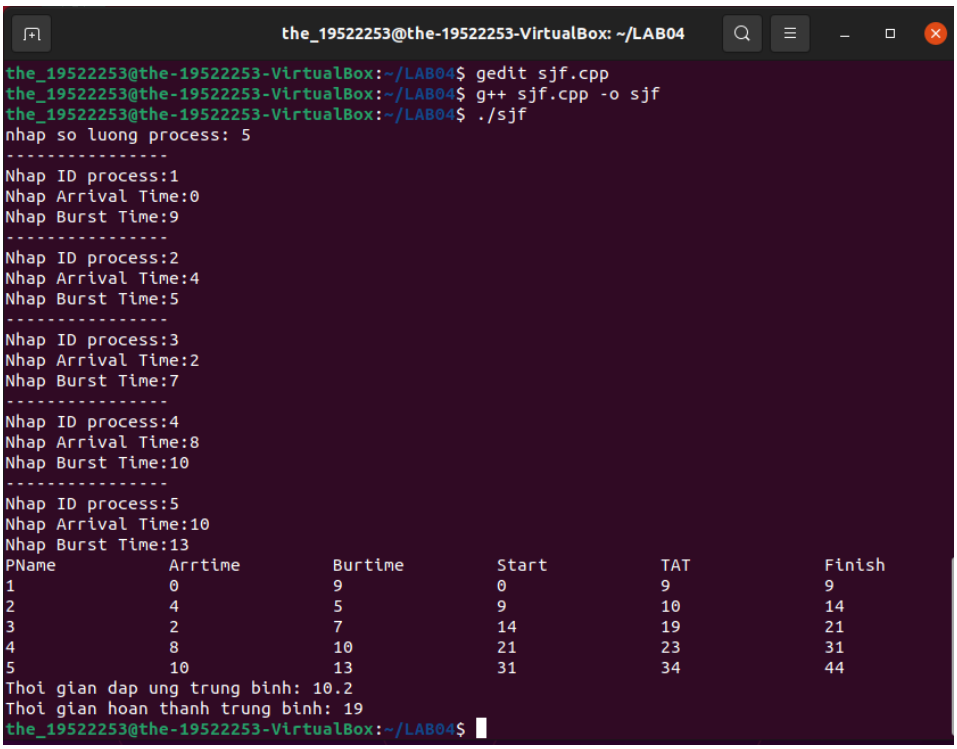
Hình 5: Code từ dòng 49 - 88

1.5. Test case

– Ví dụ 1:

Process 1	Arrival Time	Burst Time
P1	0	8
P2	4	5
P3	2	7
P4	8	10
P5	10	13

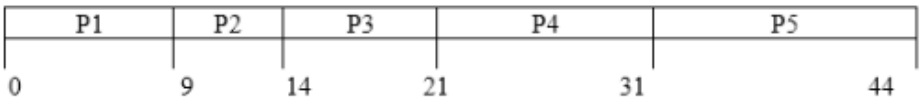
- Kết quả khi chạy code



Hình 6: Kết quả khi giải ví dụ 1 bằng code giải thuật SJF

- Kết quả khi giải tay

+ Giản đồ Gantt:



+ Thời gian đáp ứng:

P1 = 0, P2 = 5, P3 = 12, P4 =13, P5 = 21

⇨ Thời gian đáp ứng trung bình: (0 + 5 + 12 + 13 + 21) / 5 = 10.2

+ Thời gian đợi:

P1 = 0, P2 = 5, P3 = 12, P4 =13, P5 = 21

⇨ Thời gian đợi trung bình: (0 + 5 + 12 + 13 + 21) / 5 = 10.2

+ Thời gian hoàn thành:

P1 = 9, P2 = 10, P3 = 19, P4 = 23, P5 = 34

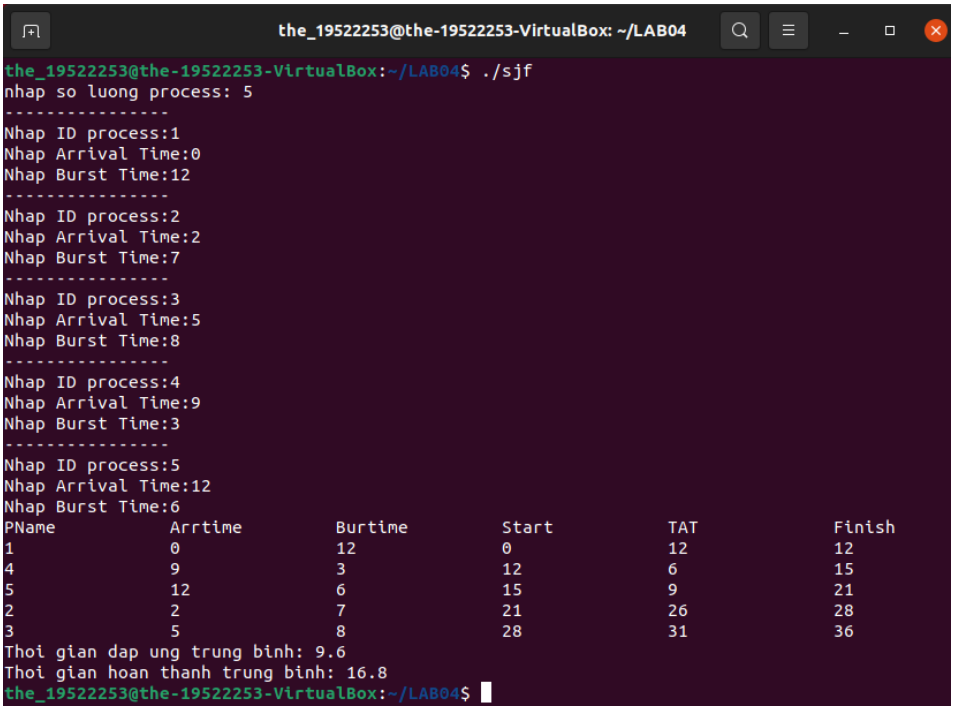
⇨ Thời gian hoàn thành trung bình: (9 + 10 + 19 + 23 + 34) / 5 = 19

Hình 7:Kết quả khi giải tay ví dụ 1 bằng giải thuật SJF

– Ví dụ 2:

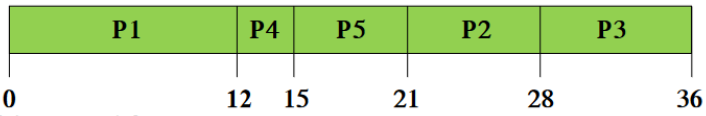
Process	Arriva Time	Burst Time
P1	0	12
P2	2	7
P3	5	8
P4	9	3
P5	12	6

- Kết quả khi chạy code



Hình 8: Kết quả khi giải ví dụ 2 bằng code giải thuật SJF

- Kết quả khi giải tay
  - Giản đồ Gantt



  - Thời gian chờ:
    - P1 = 0, P2 = 19, P3 = 23, P4 = 3, P5 = 3
    - Thời gian chờ trung bình: (0 + 19 + 23 + 3 + 3)/5 = 9.6
  - Thời gian đáp ứng:
    - P1 = 0, P2 = 19, P3 = 23, P4 = 3, P5 = 3
    - Thời gian đáp ứng trung bình: (0 + 19 + 23 + 3 + 3)/5 = 9.6
  - Thời gian hoàn thành:
    - P1 = 12, P2 = 26, P3 = 31, P4 = 6, P5 = 9
    - Thời gian hoàn thành trung bình: (12 + 26 + 31 + 6 + 9)/5 = 16.8

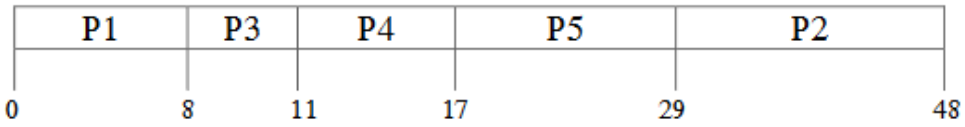
Hình 9:Kết quả khi giải tay ví dụ 2 bằng giải thuật SJF

– Ví dụ 3:

Process	Arriva Time	Burst Time
P1	0	8
P2	2	19
P3	4	3
P4	5	6
P5	7	12

- Kết quả khi giải tay

+ Giản đồ Gantt:

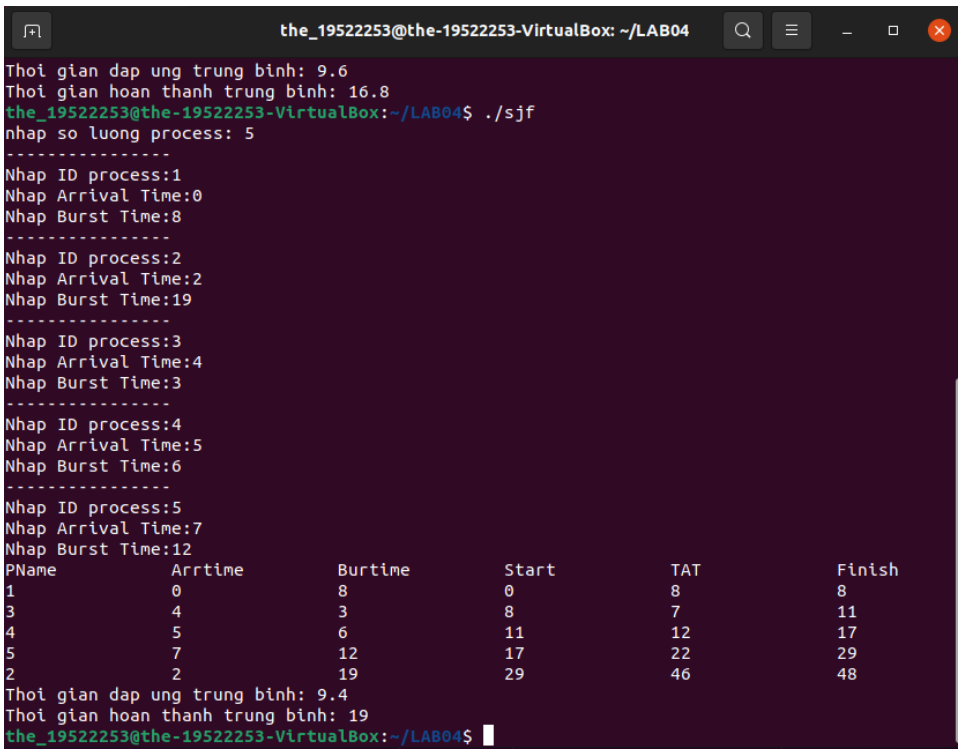


+ Thời gian đáp ứng trung bình là: 9.4

+ Thời gian hoàn thành trung bình: 19.

Hình 10: Kết quả khi giải tay ví dụ 3 bằng giải thuật SJF

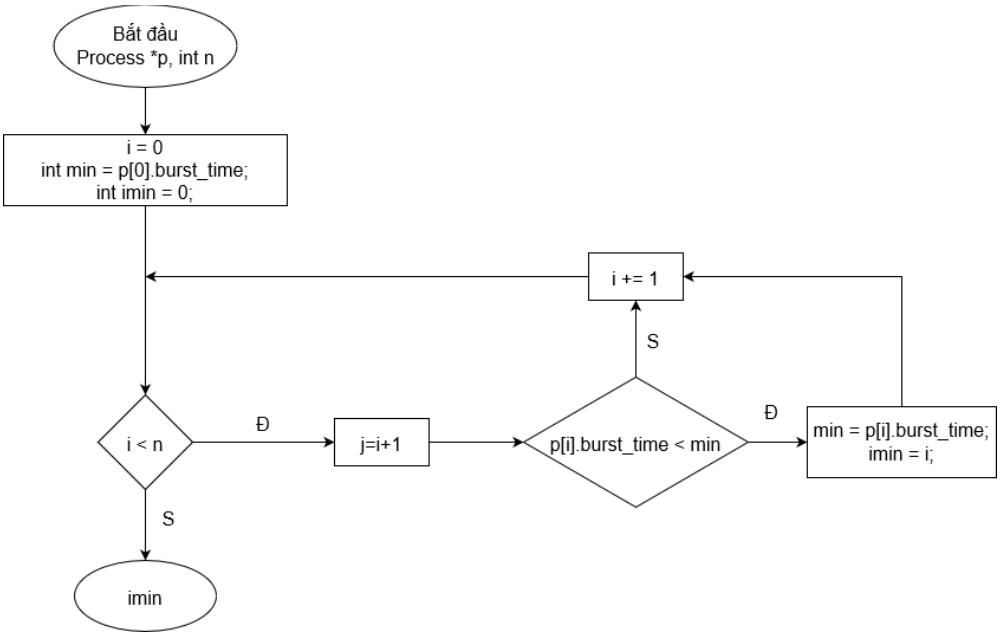
- Kết quả khi chạy code



Hình 11: Kết quả khi giải ví dụ 3 bằng code giải thuật SJF

2. Task name 2: Viết chương trình mô phỏng giải thuật SRT.

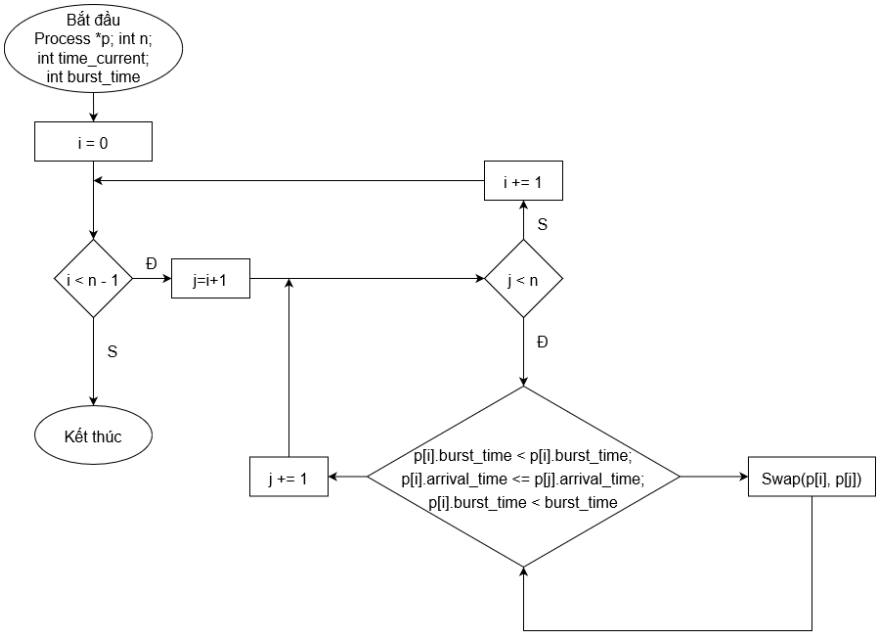
2.1. Hàm tìm ra tiến trình có burst time nhỏ nhất.



Hình 12: Lưu đồ hàm minBurstTime

- **Giải thích:** Hàm có chức năng tìm ra tiến trình có burstime nhỏ nhất bằng cách lọc qua tất cả các tiến trình trong hàng đợi.

2.2. Hàm sort các tiến tiền dựa theo tiến trình có burst_time nhỏ hơn burst của tiến trình đang thực thi.



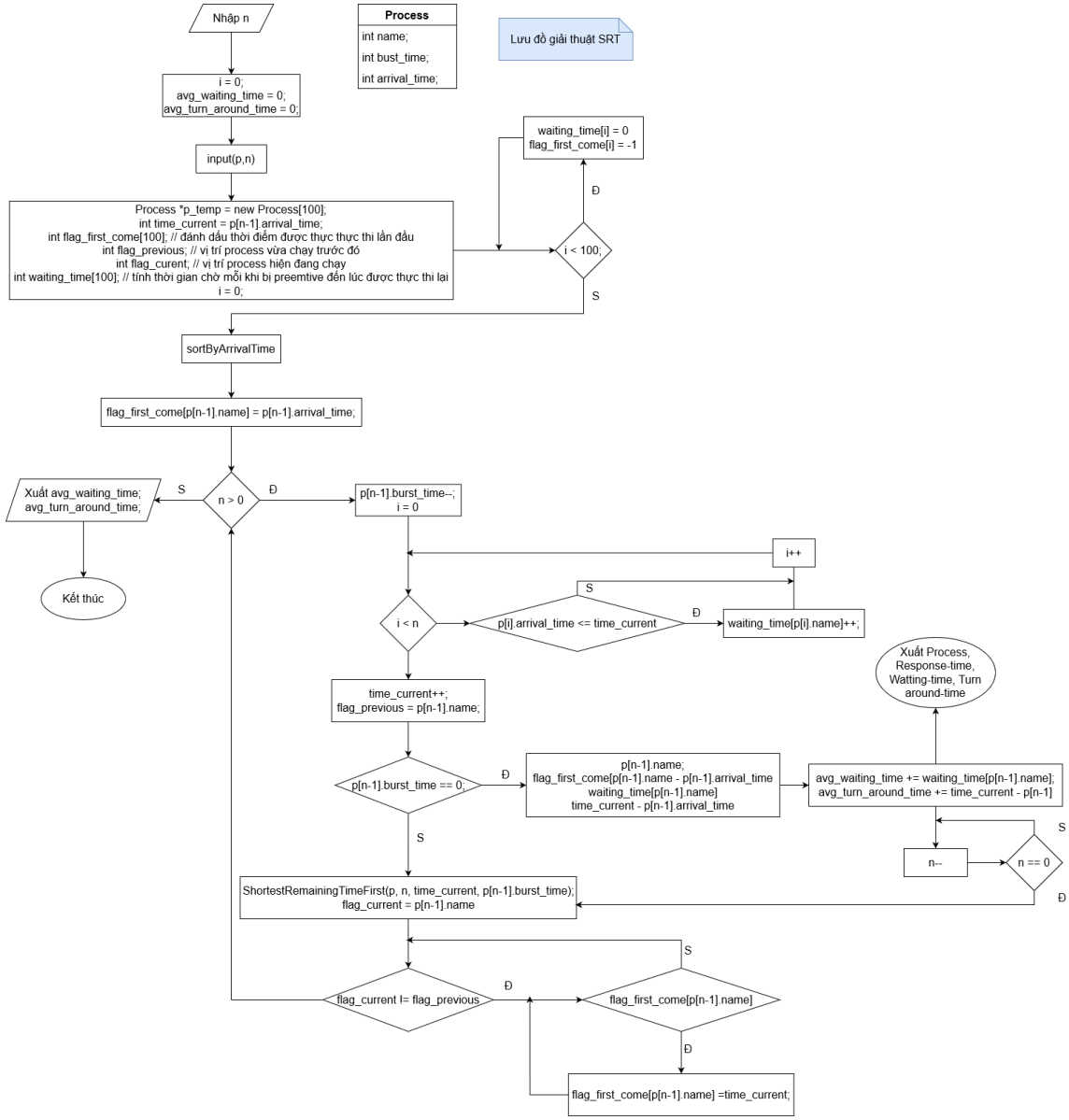
Hình 13: Hàm ShortestRemainingTimeFirst.

- **Giải thích:** Hàm dùng phương pháp nổi bọt để lọc qua các cặp tiến trình và sort các giá trị có burst_time nhỏ hơn burst time của tiến trình đang được thực thi.

2.3. Lưu đồ giải thuật SRT

- **Giải thích:**
  - Các bước đầu sẽ là tạo struct và tiến hành nhập các process tương tự như giải thuật SJF.
  - Sau đó ta sẽ có các biến như là time_current là timeline của chương trình, flag_first_com là list đánh dấu các thời điểm thực thi lần đầu.
  - flag_previous: Vị trí của process vừa chạy trước đó, lag_current: vị trí của tiến trình dang chạy; waitting_time: là thời gian chờ mỗi khi bị preemtive đến lúc được thực thi lại.
  - Ta chạy hàm for cho các mảng waiting_time và flag_first_come để đánh dấu. -1 là chỉ truy cập 1 lần.
  - Sau đó sử dụng hàm SortByArrivalTime để sort tiến trinh.
  - Duyệt từ cuối lên. Ta xép từ từ chậm rãi. Hàm for đầu tiên có tác dụng là tăng waiting time khi processs đã đến hàng đượi mà chưa được thực thi.

- Tăng timeline lên dần, và lưu tên process sắp rời đi.
- Với hàm if tiếp theo là nếu đã thực thi hết, không còn burst thì xuất trạng thái. Và ta tính các thông tin Start, TAT, End và cộng dồn thời gian chờ và thời gian hoàn thành. Sau đó giảm n-- để thu hẹp các tiến trình. Khi nào n = 0 thì thoát vòng lặp.
- Dùng Hàm ShortestRemainingTimeFirst(p, n, time_current, p[n-1] .burst_time) để chọn ra các tiến trình có burst < burst còn lại của p[flag_current].
- Hàm if ở cuối có nghĩa là nếu xảy ra trường hợp chuyển ngữ cảnh thì thì thời điểm đánh dấu sẽ bằng timeline chương trình.



Hình 14: Lưu đồ giải thuật SRT

2.4. Code giải thuật SRT

```
/*################
# University of Information Technology
# IT007 Operating System
# Pham Duc The, 19522253
# File: srt.cpp
################*/

#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 burst_remaining[100];
    int is_completed[100];
    memset(is_completed,0,sizeof(is_completed));
    cout << setprecision(2) << fixed;

    cout<<"Nhap so luong process: ";
    cin>>n;


    for (int i = 0; i < n; i++) {
                cout << "----------------" << endl;
                cout << "Nhap ID process: "; cin >> p[i].pid;
                cout << "Nhap arrival time: "; cin >> p[i].arrival_time;
```

Hình 15: Code giải thuật SRT từ dòng 1 - 50

```
47      for (int i = 0; i < n; i++) {
48                  cout << "-----------------" << endl;
49                  cout << "Nhap ID process: "; cin >> p[i].pid;
50                  cout << "Nhap arrival time: "; cin >> p[i].arrival_time;
51                  cout << "Nhap burst time: "; cin >> p[i].burst_time;
52          burst_remaining[i] = p[i].burst_time;
53          cout<<endl;
54          }
55      int current_time = 0;
56      int completed = 0;
57      int prev = 0;
58
59      while(completed != n) {
60          int idx = -1;
61          int mn = 10000000;
62          for(int i = 0; i < n; i++) {
63              if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
64                  if(burst_remaining[i] < mn) {
65                      mn = burst_remaining[i];
66                      idx = i;
67                  }
68                  if(burst_remaining[i] == mn) {
69                      if(p[i].arrival_time < p[idx].arrival_time) {
70                          mn = burst_remaining[i];
71                          idx = i;
72                          }
73                  }
74              }
75          }
76
77          if(idx != -1) {
78              if(burst_remaining[idx] == p[idx].burst_time) {
79                  p[idx].start_time = current_time;
80                  total_idle_time += p[idx].start_time - prev;
81              }
82              burst_remaining[idx] -= 1;
83              current_time++;
84              prev = current_time;
85
86              if(burst_remaining[idx] == 0) {
87                  p[idx].completion_time = current_time;
88                  p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
89                  p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
90                  p[idx].response_time = p[idx].start_time - p[idx].arrival_time;
91
92                  total_turnaround_time += p[idx].turnaround_time;
93                  total_waiting_time += p[idx].waiting_time;
94                  total_response_time += p[idx].response_time;
95
```

Hình 16: Code giải thuật SRT từ dòng 47 - 95

```
91
92                 total_turnaround_time += p[idx].turnaround_time;
93                 total_waiting_time += p[idx].waiting_time;
94                 total_response_time += p[idx].response_time;
95
96                 is_completed[idx] = 1;
97                 completed++;
98             }
99         }
100        else {
101             current_time++;
102        }
103        }
104    int min_arrival_time = 10000000;
105    int max_completion_time = -1;
106    for(int i = 0; i < n; i++) {
107        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
108        max_completion_time = max(max_completion_time,p[i].completion_time);
109    }
110
111    avg_turnaround_time = (float) total_turnaround_time / n;
112    avg_waiting_time = (float) total_waiting_time / n;
113    avg_response_time = (float) total_response_time / n;
114    cpu_utilisation = ((max_completion_time - total_idle_time) / (float)
   max_completion_time )*100;
115    throughput = float(n) / (max_completion_time - min_arrival_time);
116
117    cout<<endl<<endl;
118
119    cout<<"#pName\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;
120
121    for(int i = 0; i < n; i++) {
122 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;
123    }
124    cout<<"Thoi gian hoan thanh trung binh: = "<<avg_turnaround_time<<endl;
125    cout<<"Thoi gian dap ung trung binh: = "<<avg_waiting_time<<endl;
126
127 }
128
```

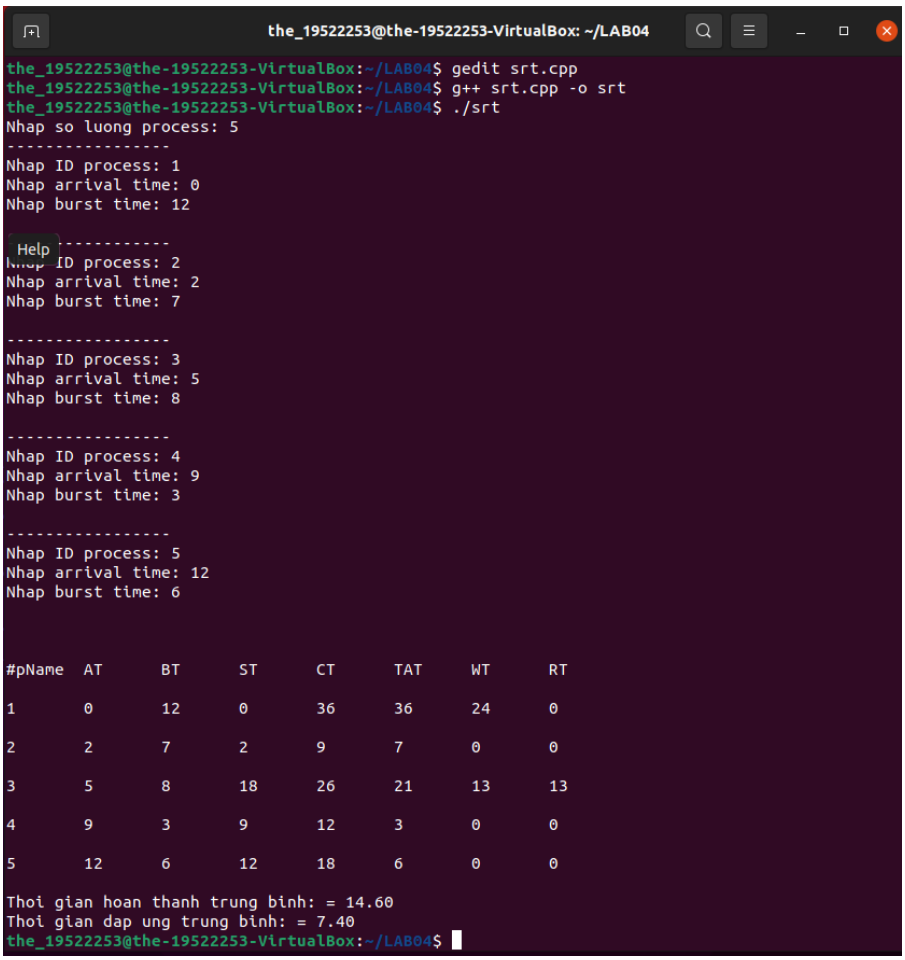
Hình 17: Code giải thuật SRT từ dòng 92 - 128

2.5. Test case

– Ví dụ 1:

Process 1	Arrival Time	Burst Time
P1	0	12
P2	2	7
P3	5	8
P4	9	3
P5	12	6

- Kết quả khi chạy code:



Hình 18: Kết quả khi giải ví dụ 1 bằng code giải thuật SRT

- Kết quả khi giải tay:
  - Giản đồ Gantt

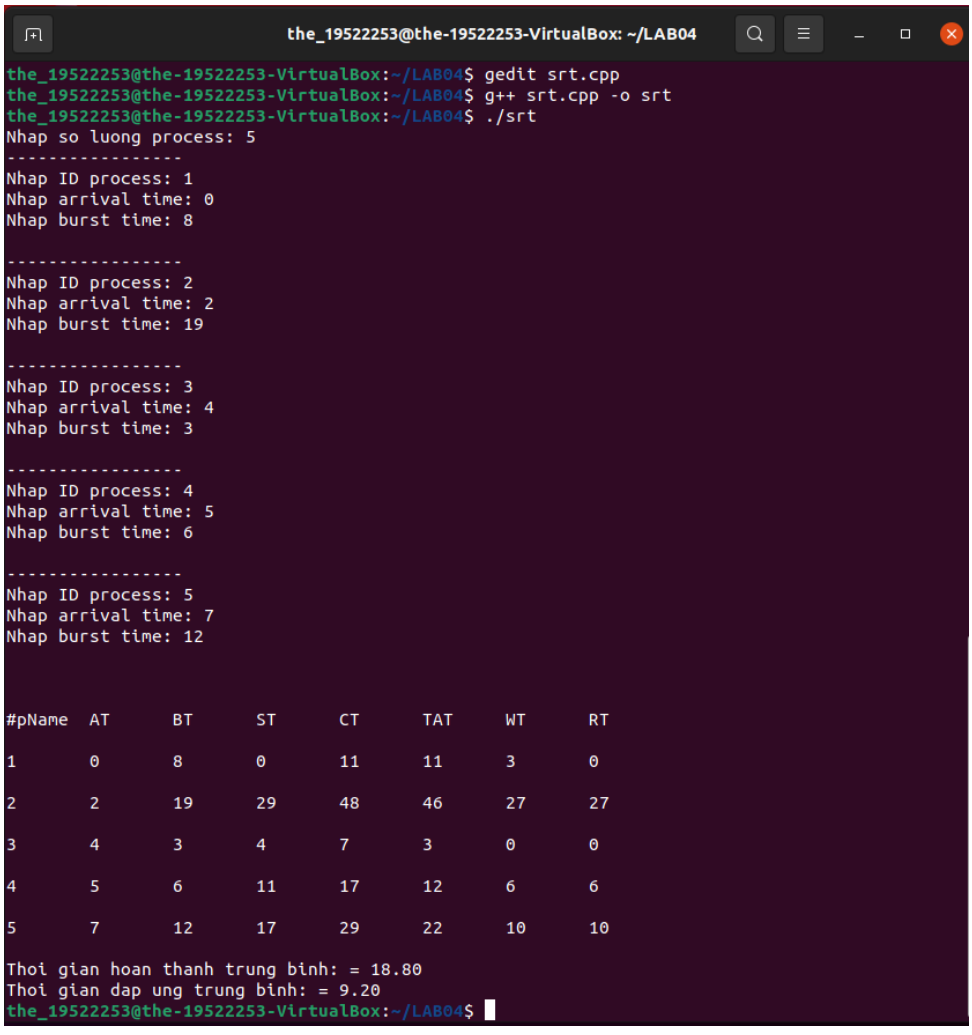
  - Thời gian chờ:
    - P1 = 24, P2 = 0, P3 = 13, P4 = 0, P5 = 0
    - Thời gian chờ trung bình: (24 + 0 + 13 + 0 + 0)/5 = 7.4
  - Thời gian đáp ứng:
    - P1 = 0, P2 = 0, P3 = 13, P4 = 0, P5 = 0
    - Thời gian đáp ứng trung bình: (0 + 0 + 13 + 0 + 0)/5 = 2.6
  - Thời gian hoàn thành:
    - P1 = 36, P2 = 7, P3 = 21, P4 = 3, P5 = 6
    - Thời gian hoàn thành trung bình: (36 + 7 + 21 + 3 + 6)/5 =14.6

Hình 19: Kết quả khi giải tay ví dụ 1 bằng giải thuật SRT

– Ví dụ 2:

Process 1	Arrival Time	Burst Time
P1	0	8
P2	2	19
P3	4	3
P4	5	6
P5	7	12

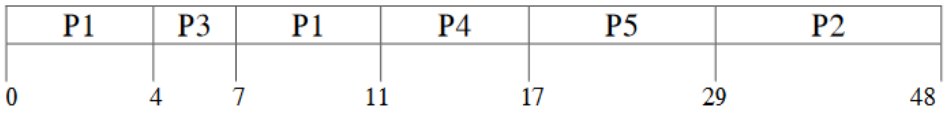
- Kết quả khi chạy code:



Hình 20: Kết quả khi giải ví dụ 2 bằng code giải thuật SRT

- Kết quả khi giải tay:

+ Giản đồ Gantt:



+ Thời gian đáp ứng trung bình là: 9.2.

+ Thời gian hoàn thành trung bình: 18.8.

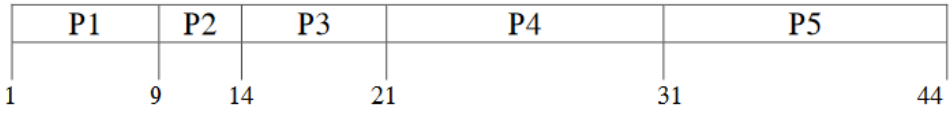
Hình 21: Kết quả khi giải tay ví dụ 2 bằng giải thuật SRT

– Ví dụ 3:

Process 1	Arrival Time	Burst Time
P1	0	9
P2	4	5
P3	2	7
P4	8	10
P5	10	13

- Kết quả khi giải tay:

+ Giản đồ Gantt:

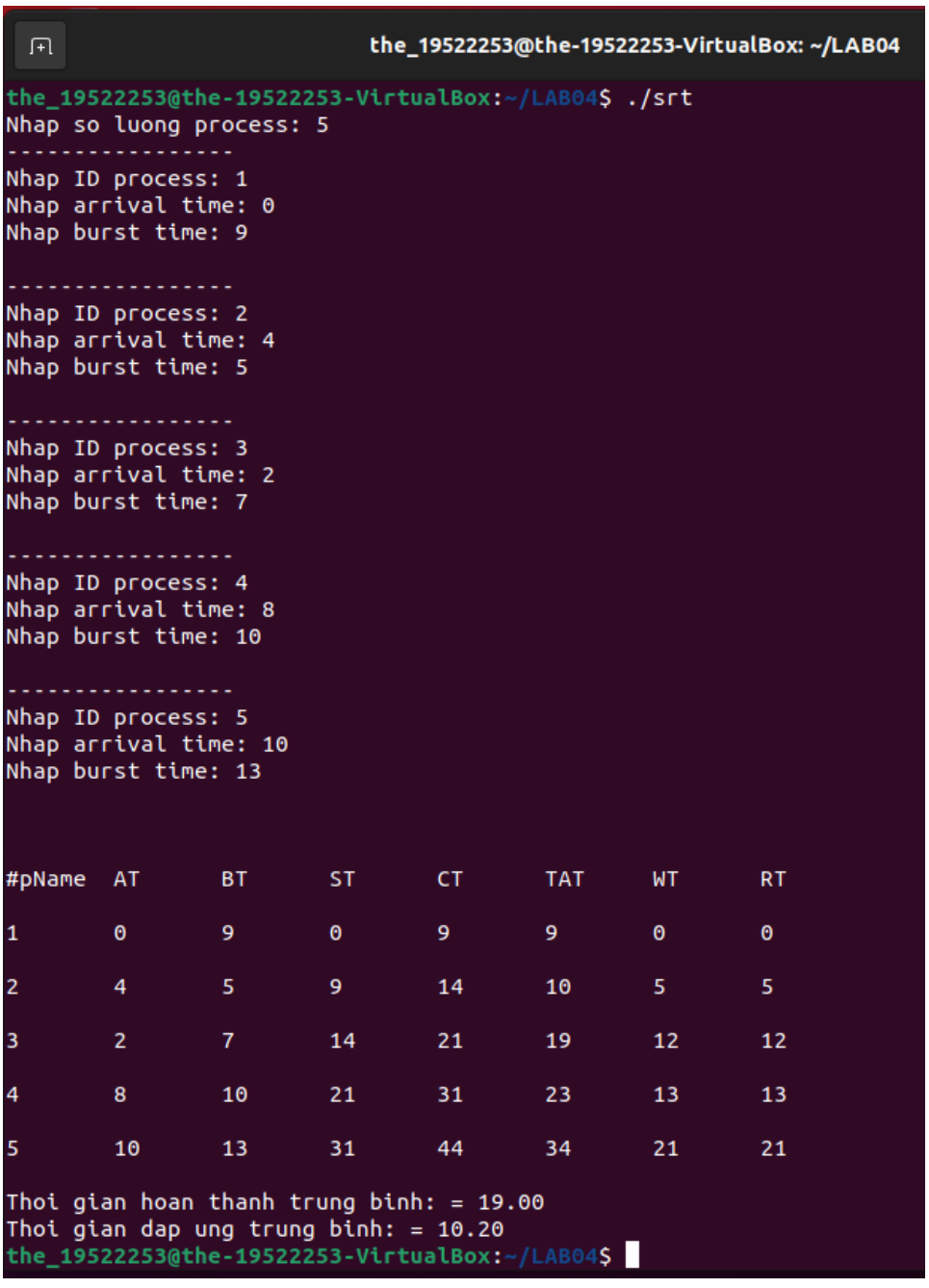


+ Thời gian đáp ứng trung bình là: 10.2

+ Thời gian hoàn thành trung bình: 19

Hình 22: Kết quả khi giải tay ví dụ 3 bằng giải thuật SRT

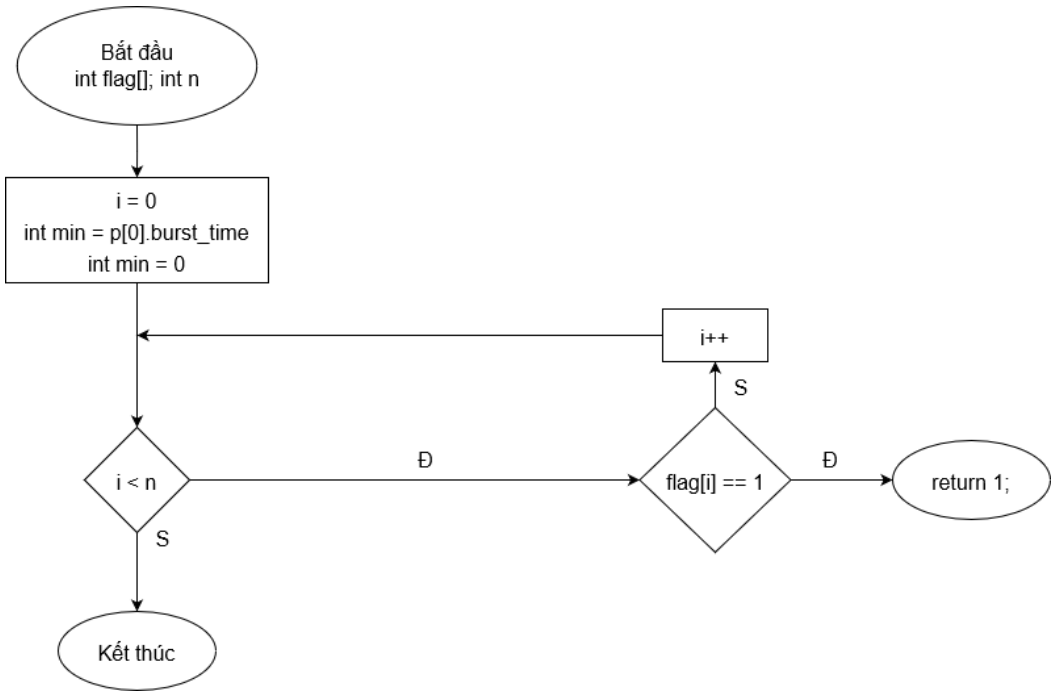
- Kết quả khi chạy code



Hình 23: Kết quả khi giải ví dụ 3 bằng code giải thuật SRT

3. Task name 3

3.1 Hàm kiểm tra



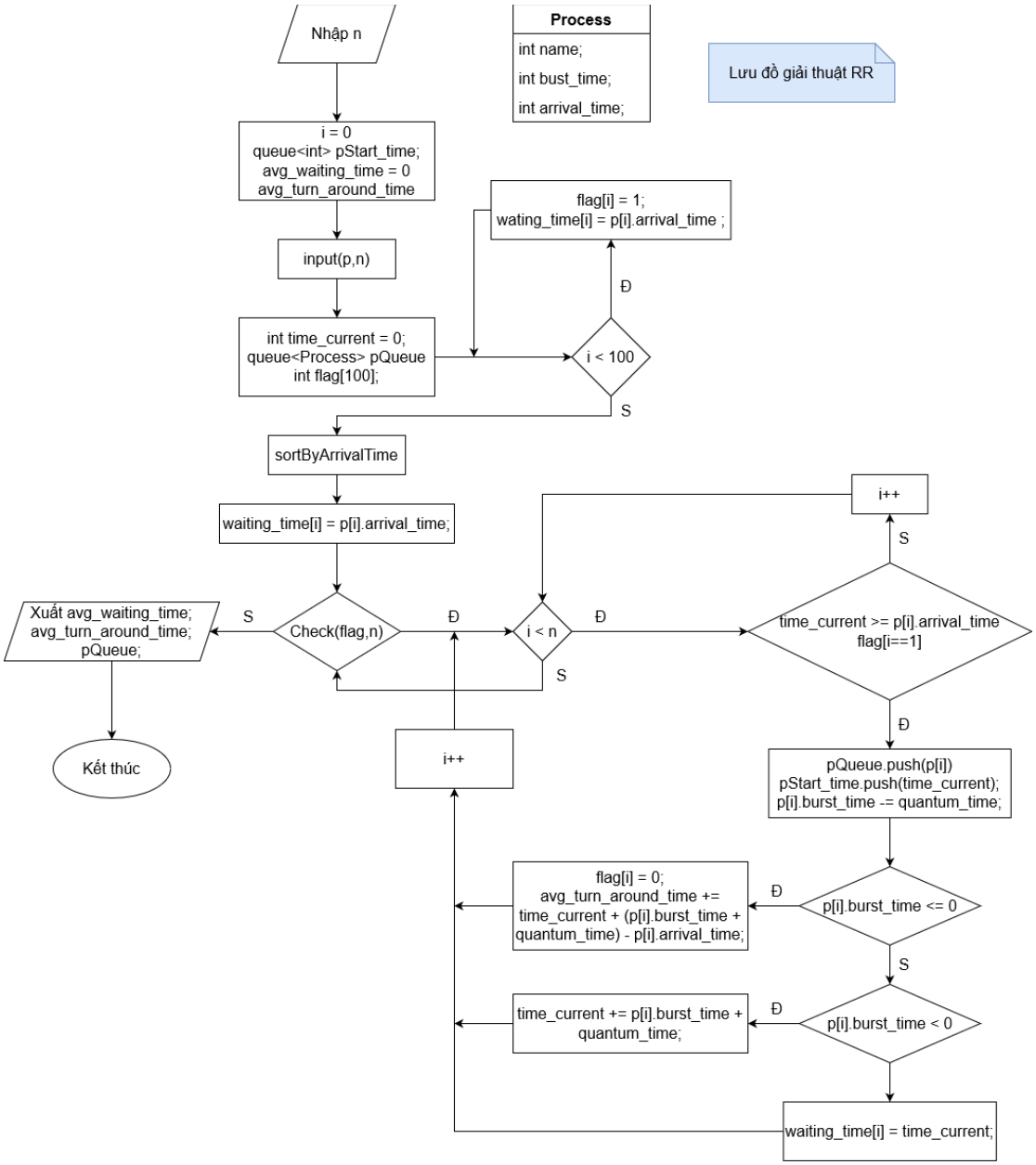
Hình 24: Hàm kiểm tra

- **Giải thích:** Hàm check có chức năng kiểm tra lại có phần tử nào trong đó bằng 1 hay ko. Nếu có thì sẽ return ra 1, còn tát cả đều bằng 0 thì return ra 0.

3.2 Giải thuật RR

- **Giải thích:**
  - Tương tự như các giải thuật trên ta sẽ tiến hành tạo struct và tiến hành nhập các thông tin. Và nhập quantumtime.
  - Tạo list flag có tác dụng kiểm tra xem busrt_time của tiến trình còn hay không. Ban đầu ta sẽ gắn hết bằng 1. Và waiting_time sẽ bằng thời gian đến.
  - Tiến hành sort theo arrival time. Và tạo ra 1 list các tiến trình rỗng khác
  - Dùng hàm check để kiểm tra xem còn tiến trình nào vẫn còn burst_time hay không.
  - Duyệt qua lần lượt tất cả các process. Tìn trình nào đã đến và còn burst time thì được xét. Ta sẽ tạo bản sao và được gắn vào list qQueue và các thông tin khác sẽ được lưu. Sau đó dảm burstTime đi với số lượng = quantum_time

- Còn nếu burst_time == 0 thì cho flag[i] =0 và tính thời gian đợi và thuời gian hoàng thành. Nếu bursttime < 0 thì thời timeline lúc này sẽ được cộng thêm burst_time và quantum_time. Các trường hợp khác thì time_current là timeline lúc này được cộng thêm quantum_time.
- Tiến hành với các tiến trình khác và kiểm tra day flag còn phần tử nào bằng 1 hay không. Nếu hết rỏi thì xuất ra qQueue. Từ đó sẽ lấy được thông tin cụ thể.



Hình 25: Lưu đồ giải thuật RR

3.3 Code giải thuật RR

```
queue<Process> SelectionFunction(Process *p, int n, int quantum_time) {
	int time_current = 0;
	int flag_c = 1;
	queue<Process> pQueue;
	int flag[100];
	sortByArrivalTime(p, n);
	for (int i = 0; i < n; i++) {
		flag[i] = 1;
		waiting_time[i] = p[i].arrival_time;
	}
	while (check(flag, n))
	{
		// Duyệt qua hết 1 lượt các process
		for (int i = 0; i < n; i++) {
			// process nào đã đến & còn burst time mới được xét
			if (time_current >= p[i].arrival_time && flag[i] == 1) {
				pQueue.push(p[i]);
				if (flag_c == 1) {
					time_current = p[i].arrival_time;
					flag_c = 0;
				}
				pStart_time.push(time_current);
				p[i].burst_time -= quantum_time;
				if (p[i].burst_time <= 0) {
					flag[i] = 0;
					ave_turnaround_time += time_current + (p[i].burst_time + quantum
				}
				ave_waiting_time += (time_current - waiting_time[i]);
				if (p[i].burst_time < 0) {
					time_current += p[i].burst_time + quantum_time;
				}
				else {
					time_current += quantum_time;
				}
				waiting_time[i] = time_current;
			}
		}
	}
```

Hình 26: Code giải thuật RR

3.4 Test case