Name: Lê Minh Nguyệt **ID:** 21521211	**Name:** Trần Trung Tín **ID:** 21522679
Name: Hồ Đức Trưởng **ID:** 21522730	**Name:** Nguyễn Công Nguyên **ID:** 21521200
Class: IT007.N11	**Group Name:** Team

OPERATING SYSTEM
LAB 04'S REPORT

SUMMARY

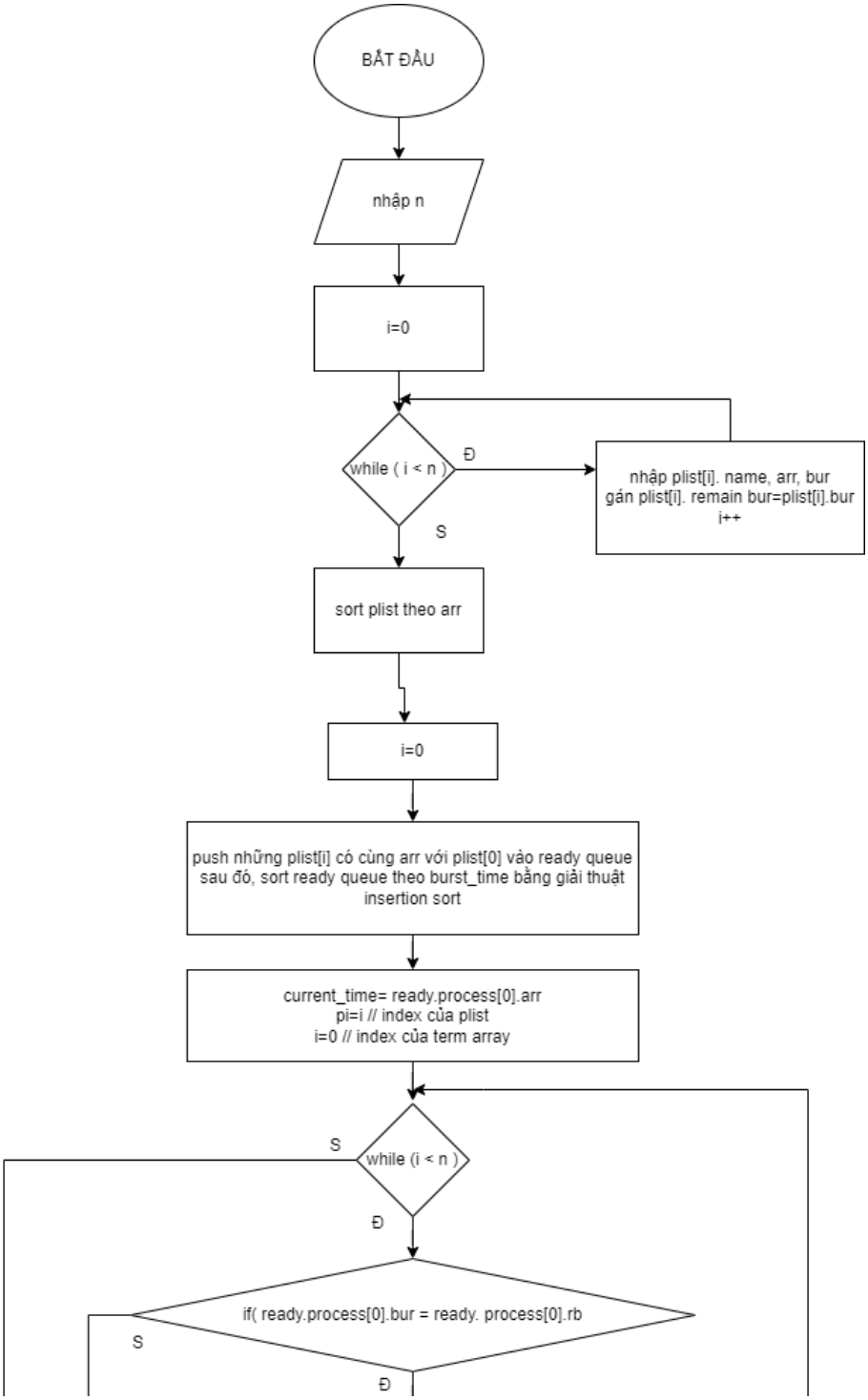
Task			Status	Page
Section 4.5	Ex 1. Giải thuật SJF	1.1. Vẽ lưu đồ giải thuật	Done	2
		1.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình		3
		1.3. Thực hiện code cho giải thuật		6
		1.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình		13
	Ex 2. Giải thuật SRTF	2.1. Vẽ lưu đồ giải thuật	Done	16
		2.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình		18
		2.3. Thực hiện code cho giải thuật		21
		2.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình		27
	Ex 3. Giải thuật RR	3.1. Vẽ lưu đồ giải thuật	Done	30
		3.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình		33
		3.3. Thực hiện code cho giải thuật		36
		3.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình		42

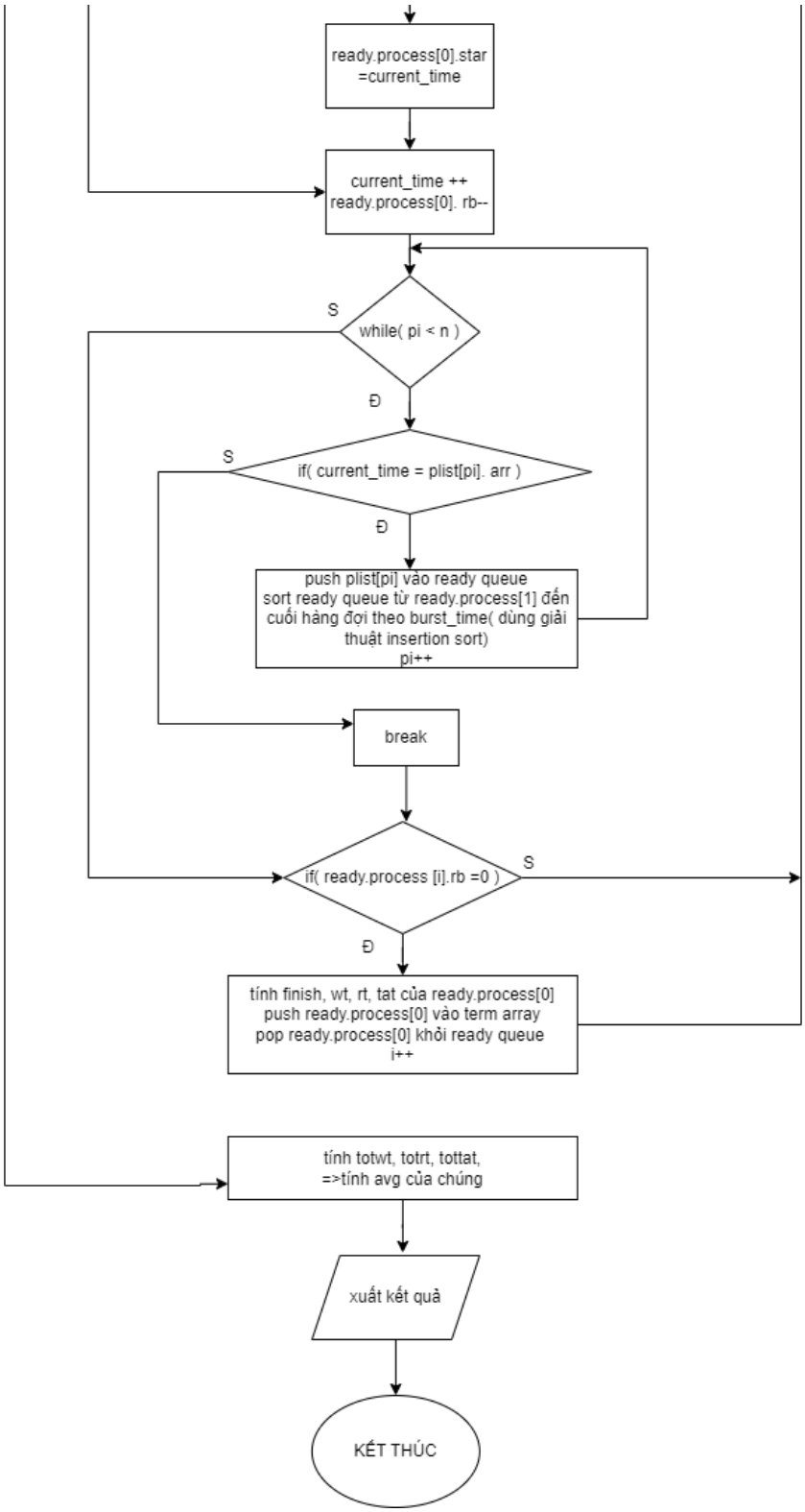
Self-scrores: 9,5

Section 4.5

Ex 1. Giải thuật SJF (Shortest Job First):

1.1. Vẽ lưu đồ giải thuật





Hình 1. Lưu đồ giải thuật SJF

1.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình

Test case: n=5

Process name	Arrival Time	Burst time
1	0	12
2	2	7
3	5	8
4	12	6
5	9	3

Trong minh họa giải thuật dưới đây, một process sẽ được trình bày các thuộc tính lần lượt theo thứ tự [process name, arrival time, burst time, remaining burst time, start, finish, wt, rt, tat]

Thực hiện chạy tay test case trên:

i. Sau quá trình nhập và khởi tạo:
   plist=[[1,0,12,12], [2,2,7,7], [3,5,8,8], [4,12,6,6], [5,9,3,3]], n=5
ii. Ta sử dụng sort theo arrvial time:
   plist=[[1,0,12,12], [2,2,7,7], [3,5,8,8], [5,9,3,3], [4,12,6,6]]
iii. Khởi tạo biến i =0, đồng thời sử dụng vòng lặp để đẩy những process có cùng arrvial time với plist[0] khi này vào hàng đợi ready mỗi lần đẩy vào ready tăng i lên 1 đơn vị.
iv. Sau đó sắp xếp hàng đợi ready theo bur theo giải thuật insertion sort.
   Ready=[[1,0,12,12]]
v. Gán current_time=ready[0].arr=0, pi=vị trí của plist chưa được push vào ready=1, i=0,
vi. Sử dụng vòng lặp khi i<5 (0<5)
   a. Tại currentime=0 kiểm ta thấy ready[0].rb==ready[0].bur
      ⇨ ready=[[1,0,12,12,0]], current_time+=1,ready[0].rb--
   b. Tại currentime=1, ready=[[1,0,12,11,0]], current_time+=1, ready[0].rb-- tại đây do ta kiểm tra thấy current_time=plist[pi].arr=2, ta sẽ push plist[1] vào ready đồng thời pi=pi+1=2,sử dụng sort(từ 1 đến vị trí cuối cùng trong hàng đợi ready) theo burst time
      ⇨ ready=[[1,0,12,10,0], [2,2,7,7]]
   c. Tương tự như vậy, tại current_time=4, ready=[[1,0,12,8,0], [2,2,7,7]], current_time+=1, ready[0].rb--, tại đây do ta kiểm tra thấy current_time=plist[pi].arr=5 ta sẽ push plist[2] vào ready đồng thời pi=pi+1=3, sử dụng sort(từ 1 đến vị trí cuối cùng trong hàng đợi ready) theo burstime
      ⇨ ready=[[1,0,12,7,0], [2,2,7,7], [3,5,8,8]]
   d. Tượng tự như vậy ⇨ current_time=8, ready=[[1,0,12,4,0], [2,2,7,7]], [3,5,8,8]], current_time+=1 ready[0].rb--, do ta kiểm tra thấy current_time=plist[pi].arr=9 ta sẽ push plist[3] vào ready đồng thời pi=pi+1=4, sử dụng sort(từ 1 đến vị trí cuối cùng trong hàng đợi ready) theo burst time ⇨ ready=[[1,0,12,3,0], [5,9,3,3], [2,2,7,7], [3,5,8,8]]

e. Tương tự như vậy đến thời điểm current_time =11, ready=[[1,0,12,1,0], [5,9,3,3], [2,2,7,7], [3,5,8,8]], current_time+=1, ready[0].rb--, do ta kiểm tra thấy current_time=plist[pi].arr=12 ta sẽ push plist[4] vào ready đồng thời pi=pi+1=5, sử dụng sort(từ 1 đến vị trí cuối cùng trong hàng đợi ready) theo burst time (do pi=5=n nên ta bỏ qua bước này trong vòng lặp
   ⇨ ready=[[1,0,12,0,0], [5,9,3,3], [4,12,6,6], [2,2,7,7], [3,5,8,8]]. Ở đây ta kiểm tra thấy ready[0].rb=0, do vậy pop ready[0] ra khỏi hàng đợi đồng thời tính toán finish, wt, rt, tat push ready[0] vào hàng đợi terminate ⇨ terminate=[[1,0,12,0,0,12,0,0,12]], i=i+1=1, ready=[[5,9,3,3], [4,12,6,6], [[2,2,7,7]], [3,5,8,8]].
f. Tại thời thời điểm current_time=12, kiểm tra thấy ready[0].bur==ready[0]rb, ready=[[5,9,3,3,12], [4,12,6,6], [2,2,7,7], [3,5,8,8]], current_time+=1, ready[0].rb--
g. Tương tự như vậy đến thời điểm current_time=14, ready=[[5,9,3,1,12], [4,12,6,6], [2,2,7,7], [3,5,8,8]], current_time+=1, ready[0].rb--.
   Ở đây ta kiểm tra thấy ready[0].rb=0 ,do vậy pop ready[0] ra khỏi hàng đợi đồng thời tính toán finish, wt, rt, tat push ready[0] vào hàng đợi terminate
   ⇨ terminate=[[1,0,12,0,0,12,0,0,12]], [5,9,3,0,12,15,3,3,6]], i=i+1=2
   Ready=[[4,12,6,6], [2,2,7,7], [3,5,8,8]].
h. Tại thời điểm current_time=15 ,kiểm tra thấy ready[0].bur==ready[0].rb ⇨ ready=[[4,12,6,6,15], [2,2,7,7], [3,5,8,8]], current_time+=1, ready[0].rb--
i. Tại thời điểm current_time=20, ready=[[4,12,6,1,15], [2,2,7,7], [3,5,8,8]], current_time+=1, ready[0].rb--.
   Ở đây ta kiểm tra thấy ready[0].rb=0 ,do vậy pop ready[0] ra khỏi hàng đợi ready đồng thời tính toán fininsh, wt, rt, tat push ready[0] vào hàng đợi terminate

   ⇨ terminate=[[1,0,12,0,0,12,0,0,12], [5,9,3,0,12,15,3,3,6], [4,12,6,0,15,21,3,3,9]], i=i+1=3

   ready=[[2,2,7,7], [3,5,8,8]].
j. Tại thời điểm current_time=21, kiểm tra thấy ready[0].bur==ready[0].rb ⇨ ready=[[2,2,7,7,21], [3,5,8,8]] current_time+=1, ready[0].rb--
k. Tại thời điểm currentime=27, ready=[[2,2,7,1,21], [3,5,8,8]] current_time+=1. Ở đây ta kiểm tra thấy ready[0].rb=0, do vậy pop ready[0] ra khỏi hàng đợi đồng thời tính toán fininsh, wt, rt, tat push ready[0] vào hàng đợi terminate
   ⇨ terminate=[[1,0,12,0,0,12,0,0,12], [5,9,3,0,12,15,3,3,6], [4,12,6,1,15,21,3,3,9], [2,2,7,0,21,28,19,19,26]], i=i+1=4
   Ready=[[3,5,8,8]].

l. Tại thời điểm current_time=28, kiểm tra thấy ready[0].bur==ready[0].rb => ready=[[3,5,8,8,28]], current_time+=1, ready[0].rb--

m. Tại thời điểm current_time=35, ready=[[3,5,8,1,28]], current_time+=1, ready[0].rb--. Ở đây ta kiểm tra thấy ready[0].rb=0 ,do vậy pop ready[0] ra khỏi hàng đợi đồng thời tính toán fininsh, wt, rt, tat push ready[0] vào hàng đợi terminate
⇨ terminate=[[1,0,12,0,0,12,0,0,12], [5,9,3,0,12,15,3,3,6], [4,12,6,0,15,21,3,3,9], [2,2,7,0,21,28,19,19,26], [3,5,8,0,28,36,23,23,31]], i=i+1=5
Ready=[]

vii. Ta kiểm tra i==n (5==5) thoát khỏi vòng lặp đồng thời tính avewt=9.6, avert=9.6, avetat=16.8

viii. Xuất kết quả ra ngoài màn hình

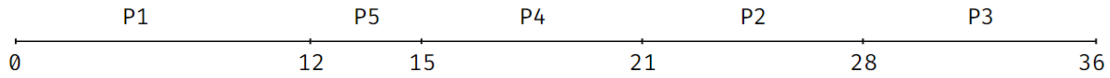
ix. Kết thúc

Kết quả sau khi chạy tay (lưu trong term[]):

Process name	Arrival time	Burst time	Remaining burst	Start	Finish	WT	RT	TAT
1	0	12	0	0	12	0	0	12
5	9	3	0	12	15	3	3	6
4	12	6	0	15	21	3	3	9
2	2	7	0	21	28	19	19	26
3	5	8	0	28	36	23	23	31

avewt=9.6, avert=9.6, avetat=16.8

So sánh với kết quả giải tay test case trên (thứ tự process name lần lượt từ 1 đến 5):



start[ ]= { 0, 21, 28, 15, 12 }
fisnish[ ] ={12, 28, 36, 21, 15}
rt[ ]={ 0, 19, 23, 3, 3 }
⇨ avert= 9.6
wt[ ]={ 0, 19, 23, 3, 3 }
⇨ avewt= 9.6
tat [ ]={ 12,26, 31, 9, 6 }
⇨ avert= 16.8

➢ ***Có thể thấy, kết quả thực hiện chạy tay trên lưu đồ giống với kết quả giải tay. Vì vậy, ta kết luận lưu đồ đúng.***

1.3. Thực hiện code cho giải thuật

```c
#include <stdio.h>

/*
Structure P contains the information of a process,
including process name, arrival time, burst time, start time, finish time,
response time, waiting time, turn-around time, and remaining burst time, respec
tively.

The pointers *print and *set point to the printP(used to print the information
of a process)
and setP function(used to calculate its rt, wt and tat).
*/
struct P {
        int pn, arr, bur, star, finish, rt, wt, tat, rb;
        void (*print)(const struct P*);
        void (*set)(struct P*);
};

/*
Structure ReadyQueque contains an array of processes and its size(the number of
 processes in the array).
*/
struct ReadyQueue {
    int size;
    struct P processes[10];
};
```

Hình 2. Tạo cấu trúc Process và Ready Queue trong giải thuật SJF

- Struct P chứa những thông tin của một process, gồm: tên, arrival time, burst time, start time, finish time, thời gian đáp ứng, thời gian đợi, thời gian hoàn thành và burst time còn lại (remaining burst time). Hai con trỏ *print và *set lần lượt thông qua hàm printP để xuất thông tin của một process ra màn hình, và setP để tính toán RT, WT, TAT.
- Struct ReadyQueue gồm một mảng process và kích cỡ của hàng đợi.

```c
/* The init() function to initialize the ready queue with size = 0.*/
void init(struct ReadyQueue *ready) {
        ready->size=0;
}

/* The setP() function is used to calculate rt, wt and tat of a process p1.*/
void setP(struct P *p1) {
        p1->rt=p1->star-p1->arr;
        p1->tat=p1->finish-p1->arr;
        p1->wt=p1->tat-p1->bur;
}

/* The printP() function is used to print the information of a process p1.*/
void printP(const struct P *p1) {
        printf("%d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\n", p1->pn, p1->arr, p1->
bur, p1->star, p1->finish, p1->rt, p1->wt, p1->tat);
}
```

Hình 3. Hàm khởi tạo Ready Queue, hàm tính RT, WT, TAT và hàm in Process trong giải thuật SJF

- Hàm init() khởi tạo hàng đợi Ready ban đầu không có phần tử nào.
- Các hàm setP() và printP() thực hiện tính toán và xuất kết quả như hình 3.

```
/*
The SortArr() function is used to sort a process array plist[] with n elements
in ascending order by arrival time.
*/
void SortArr(struct P plist[], int n) {
        int i, j;
        for(i=0; i<n; i++) {
                for(j=i+1; j<n; j++) {
                        if(plist[i].arr>plist[j].arr) {
                                struct P P0;
                                P0=plist[i];
                                plist[i]=plist[j];
                                plist[j]=P0;
                        }
                }
        }
}
```

Hình 4. Hàm sắp xếp Process theo arrival time tăng dần trong giải thuật SJF

```
/*
The SortBur() function is used to sort a process array plist[] with n elements
in ascending order by burst time.

** This function should use Insertion Sort Algorithm,
so that a new-comming-to-ready-queue process will stand right behind the one
that has the same burst time as it but has arrived before
(in case there're processes having the same burst time,
the one coming first will run first).
*/
void SortBur(struct P plist[], int l, int n) {
        int i, j;
        struct P key;
        for (i = l+1; i < n; i++) {
        key = plist[i];
        j = i - 1;
        while (j >= l && plist[j].bur > key.bur) {
            plist[j + 1] = plist[j];
            j = j - 1;
        }
        plist[j + 1] = key;
    }
}
```

Hình 5. Hàm sắp xếp Process theo burst time tăng dần trong giải thuật SJF

- Hàm SortBur() sắp xếp một mảng process tăng dần theo burst time. Hàm này sử dụng giải thuật Insertion Sort thay vì sắp xếp thông thường, để khi một process xuất hiện mà có cùng burst time với một process khác đã ở sẵn trong hàng đợi, nó sẽ không tranh mất vị trí của process đến trước mà đứng đợi ngay sau process đến trước.

```c
/* The pushP() function pushes a process into the ready queue at the very back
of the queue.*/
void pushP(struct ReadyQueue *ready, struct P p1) {
        ready->processes[ready->size++]=p1;
}

/*
The popP() function pops the first process, which is done running, out of the r
eady queue.
The next process in the queue becomes the first one and soon starts running.
*/
void popP(struct ReadyQueue *ready) {
    for(int i=0; i<ready->size-1; i++)
        ready->processes[i]=ready->processes[i+1];
    ready->size--;
}
```

Hình 6. Hàm thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready trong giải thuật SJF

- Hàm pushP() và popP() lần lượt thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready, khi đó kích cỡ của hàng đợi cũng sẽ tăng hoặc giảm đi một phần tử tương ứng.

```c
int main() {

        /* Define a process array, called plist[]. */
        struct P plist[10];

        /* Define and initialize a ready queue. */
        struct ReadyQueue ready;
        init(&ready);

        /* Define a terminated array, which will contains all the done running
processes. */
        struct P term[10];

        int i, n;

        /* Total waiting time, total turn-around time, total response time of t
he processes. */
        int totwt=0, tottat=0, totrt=0;
```

Hình 7. Khởi tạo mảng process input, hàng đợi Ready, mảng terminated và các biến để tính tổng RT, WT, TAT trong giải thuật SJF

```
        /*** GET THE INPUT ***/
/* Input of a process includes process name, arrival time and burst time. */
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        for(i=0; i<n; i++) {
                printf("Enter the Process Name, Arrival Time & Burst Time: ");
                scanf("%d%d%d", &plist[i].pn, &plist[i].arr, &plist[i].bur);
                plist[i].print=printP;
                plist[i].set=setP;
                /* rb has inital value the same as burst time value. */
                plist[i].rb=plist[i].bur;
        }

        /* Sort the input in ascending order by arrival time. */
        SortArr(plist, n);
        /* plist[] now contains the processes with increasing arrival time
        (the same arrival times are not sorted). */
```

Hình 8. Nhập input, khởi tạo giá trị ban đầu cho rb và sắp xếp mảng process input theo arrival time

- Sau khi khởi tạo các mảng và hàng đợi, ta tiến hành nhập input gồm số lượng process n, tên, arrival time và burst time của từng process. Ta khởi tạo giá trị ban đầu cho rb (remaining burst time) bằng burst time của chính process đó.
- Các process nhập vào sẽ được lưu trong mảng plist, ta sắp xếp mảng này tăng dần theo arrival time.

```
        /*
        Push the process which arrives earliest into the ready queue.

        Use the while loop to push all the processes having the same smallest a
rrival time
        into ready queue and sort them by burst time.
        */
        i=0;
        while(plist[i].arr==plist[0].arr) {
                pushP(&ready, plist[i]);
                SortBur(ready.processes, 0, ready.size);
                i++;
    }
```

Hình 9. Đưa tất cả những process cùng xuất hiện sớm nhất vào hàng đợi Ready

- Sử dụng vòng lặp while để push tất cả những process có cùng arrival time với process đến sớm nhất vào hàng đợi Ready. Sau đó, ta tiến hành sắp xếp toàn bộ hàng đợi Ready theo burst time (bắt đầu từ vị trí 0 đến cuối hàng đợi).
- Sau khi push xong những process này, i sẽ chỉ mục đến process có arrival time nhỏ thứ 2 trong plist.

```
/* The variable current_time simulates how the ready queue works in real time.
current_time has initial value of arrival time of the first running process. */
    int current_time = ready.processes[0].arr;

/* pi indexes to the processes in plist[] that haven't been pushed to ready que
ue yet. */
    int pi=i;

/* A process when popped out of ready queue will be added to term[] array at th
e index i, starting at 0. */
    i=0;

/* Keep doing these handling works until all the processes are added to term[].
 */
    while(i<n) {

/* If a process runs for the first time, its start time is current_time. */
        if (ready.processes[0].bur == ready.processes[0].rb)
                        ready.processes[0].star = current_time;

/* When a time unit passes, remaining burst time of running process is reduced
by 1. */
        current_time++;
        ready.processes[0].rb--;
```

Hình 10. Khởi tạo giá trị ban đầu cho current_time, pi, i và tính start time cho process đang chạy

- Ta giả định hàng đợi Ready là một mảng các process, và phần tử đầu tiên trong mảng này luôn là process đang chạy.
- Biến current_time chứa giá trị thời gian hiện tại, dùng để mô phỏng cách hàng đợi Ready làm việc trong thời gian thực. Ta khởi tạo giá trị ban đầu của current_time là thời điểm xuất hiện của process đầu tiên được chọn để thực thi (process đến sớm nhất và có burst time nhỏ nhất so với các process đến cùng lúc).
- Biến pi chỉ mục đến các process trong plist chưa được thêm vào hàng đợi Ready (vì chúng chưa xuất hiện). pi bắt đầu từ vị trí i.
- Biến i đặt lại bằng 0, chỉ mục đến các process trong mảng term[] (là mảng chứa các process đã hoàn thành thực thi).
- Ta bắt đầu xử lý định thời CPU, dừng lại khi tất cả các process trong plist đều đã được thêm vào term[].
- Tại thời điểm current_time, nếu process đầu tiên trong hàng đợi có bur bằng rb (tức nó chưa ở trong trạng thái running trước đó), current_time chính là start time của nó.
- current_time tăng thêm 1, cùng lúc burst time của process đang chạy giảm đi 1.

```
/* Push all the processes arriving at current_time to ready queue and sort the
queue by burst time.
Only do this if there's any process in plist[] not been pushed into ready queue
 yet. */
        while (pi<n) {
                if(current_time==plist[pi].arr) {
                                pushP(&ready, plist[pi]);
                                SortBur(ready.processes, 1, ready.size);
                                pi++;
                        }
/* If at current_time, there's no process arriving, get out of the loop
to continue counting current_time. */
                else break;
        }

/* If remaining burst time of the running process equals 0, it finishes at curr
ent_time.
So, we calculate its rt, wt, and tat via set function.
Then add it to term[] at the index i (increase i by 1 after that), and finally
pop it out of the queue. */
        if(ready.processes[0].rb==0) {
            ready.processes[0].finish=current_time;
            ready.processes[0].set(&ready.processes[0]);
            term[i++]=ready.processes[0];
            popP(&ready);
        }
    }
```

Hình 11. Đưa process vừa xuất hiện vào hàng đợi Ready và sắp xếp, thực hiện các thao tác đối với process đang chạy nếu nó hoàn thành thực thi

- Tại thời điểm current_time mới này, ta sử dụng vòng lặp while để thêm tất cả những process đến tại thời điểm này (arrival time == current_time) vào hàng đợi Ready và tiến hành sắp xếp lại hàng đợi theo burst time của chúng. Ta *chỉ thực hiện sắp xếp từ vị trí 1 đến cuối hàng đợi, vì process đầu tiên vẫn đang thực thi và sẽ không bị ngắt cho đến khi hoàn thành trong giải thuật SJF.*
- Nếu tại current_time, không có process nào đến, ta thoát khỏi vòng lặp để tiếp tục đếm current_time.
- Cũng tại thời điểm này, nếu process đang chạy có rb == 0 (tức nó đã chạy xong), finish time của nó chính bằng thời điểm hiện tại. Ta thực hiện tính RT, WT, TAT cho nó thông qua hàm set() và thêm nó vào vị trí i của mảng term[]. Sau đó, ta tăng i lên 1 và xóa process đó ra khỏi hàng đợi Ready. Lúc này, process tiếp theo sẽ được đẩy lên vị trị đầu tiên trong hàng đợi và thực thi.

```
/* Print the information of the terminated processes by their finishing order.
Calculate the total numbers. */
        printf("\nPName\tArrtime\tBurtime\tStart\tFinish\tRT\tWT\tTAT\n");
        for(i=0; i<n; i++) {
                term[i].print(&term[i]);
                totwt+=term[i].wt;
                tottat+=term[i].tat;
                totrt+=term[i].rt;
        }

/* Calculate the average numbers and print them. */
        float avewt, avetat, avert;
        avewt=(float) totwt/n;
        avetat=(float) tottat/n;
        avert=(float) totrt/n;
        printf("Average response time: %0.1f\nAverage waiting time: %0.1f\nAver
age turnaround time: %0.1f\n", avert, avewt, avetat);
}
                                                              212,1         Bot
```

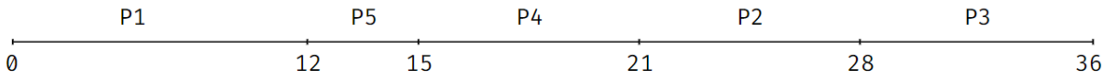
Hình 12. Thực hiện tính toán RT, WT, TAT trung bình và xuất kết quả ra màn hình

1.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình

Test case 1: n=5

Process name	Arrival Time	Burst time
1	0	12
2	2	7
3	5	8
4	12	6
5	9	3

Kết quả giải tay:



Hình 13. Giản đồ Gantt cho test case 1 giải thuật SJF

start[ ]= { 0, 21, 28, 15, 12 }
fisnish[ ] ={12, 28, 36, 21, 15}
rt[ ]={ 0, 19, 23, 3, 3 }
⇨ avert= 9.6
wt[ ]={ 0, 19, 23, 3, 3 }
⇨ avewt= 9.6
tat [ ]={ 12,26, 31, 9, 6 }

⇨ avert= 16.8

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ vim sjf.c
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ gcc sjf.c -o sjf
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./sjf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 12
Enter the Process Name, Arrival Time & Burst Time: 2 2 7
Enter the Process Name, Arrival Time & Burst Time: 3 5 8
Enter the Process Name, Arrival Time & Burst Time: 4 12 6
Enter the Process Name, Arrival Time & Burst Time: 5 9 3

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
1            0      12       0      12       0       0      12
5            9       3      12      15       3       3       6
4           12       6      15      21       3       3       9
2            2       7      21      28      19      19      26
3            5       8      28      36      23      23      31
Average response time: 9.6
Average waiting time: 9.6
Average turnaround time: 16.8
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

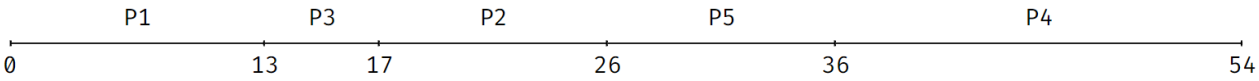
Hình 14. Chạy test case 1 bằng code giải thuật SJF

➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 2: n=5

Process name	Arrival Time	Burst time
1	0	13
2	4	9
3	6	4
4	7	18
5	12	10

Kết quả giải tay:



Hình 15. Giản đồ Gantt cho test case 2 giải thuật SJF

start[ ]= { 0, 17, 13, 36, 26 }
fisnish[ ] ={13,26,17, 54, 36}
rt[ ]={ 0, 13,7, 29, 14 }
⇨ avert= 12.6
wt[ ]={ 0, 13,7, 29, 14 }

⇨ avert= 12.6
tat [ ]={ 13, 22, 11, 47, 24 }
⇨ avert= 23.4

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./sjf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 13
Enter the Process Name, Arrival Time & Burst Time: 2 4 9
Enter the Process Name, Arrival Time & Burst Time: 3 6 4
Enter the Process Name, Arrival Time & Burst Time: 4 7 18
Enter the Process Name, Arrival Time & Burst Time: 5 12 10

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
1            0      13       0      13       0       0      13
3            6       4      13      17       7       7      11
2            4       9      17      26      13      13      22
5           12      10      26      36      14      14      24
4            7      18      36      54      29      29      47
Average response time: 12.6
Average waiting time: 12.6
Average turnaround time: 23.4
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

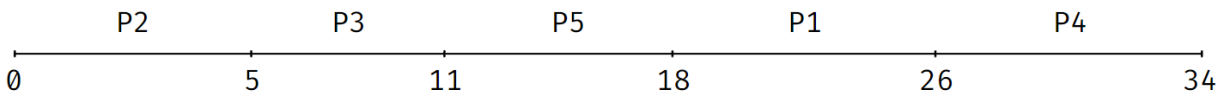
Hình 16. Chạy test case 2 bằng code giải thuật SJF

➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 3: n=5

Process name	Arrival Time	Burst time
1	0	8
2	0	5
3	2	6
4	2	8
5	4	7

Kết quả giải tay:



Hình 17. Giản đồ Gantt cho test case 3 giải thuật SJF

start[ ]= { 18, 0, 5, 26, 11}

fisnish[ ] ={26, 5, 11 , 34, 18 }
rt[ ]={ 18, 0, 3, 24, 7 }

⇨ avert= 10.4

wt[ ]={ 18, 0, 3, 24, 7 }

⇨ avert= 10.4

tat [ ]={ 26, 5, 9, 32, 14}

⇨ avert= 17.2

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./sjf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 8
Enter the Process Name, Arrival Time & Burst Time: 2 0 5
Enter the Process Name, Arrival Time & Burst Time: 3 2 6
Enter the Process Name, Arrival Time & Burst Time: 4 2 8
Enter the Process Name, Arrival Time & Burst Time: 5 4 7

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
2            0       5       0       5       0       0       5
3            2       6       5       11      3       3       9
5            4       7       11      18      7       7       14
1            0       8       18      26      18      18      26
4            2       8       26      34      24      24      32
Average response time: 10.4
Average waiting time: 10.4
Average turnaround time: 17.2
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

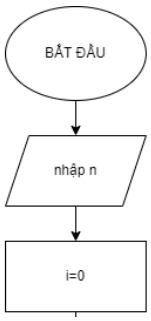
Hình 18. Chạy test case 3 bằng code giải thuật SJF

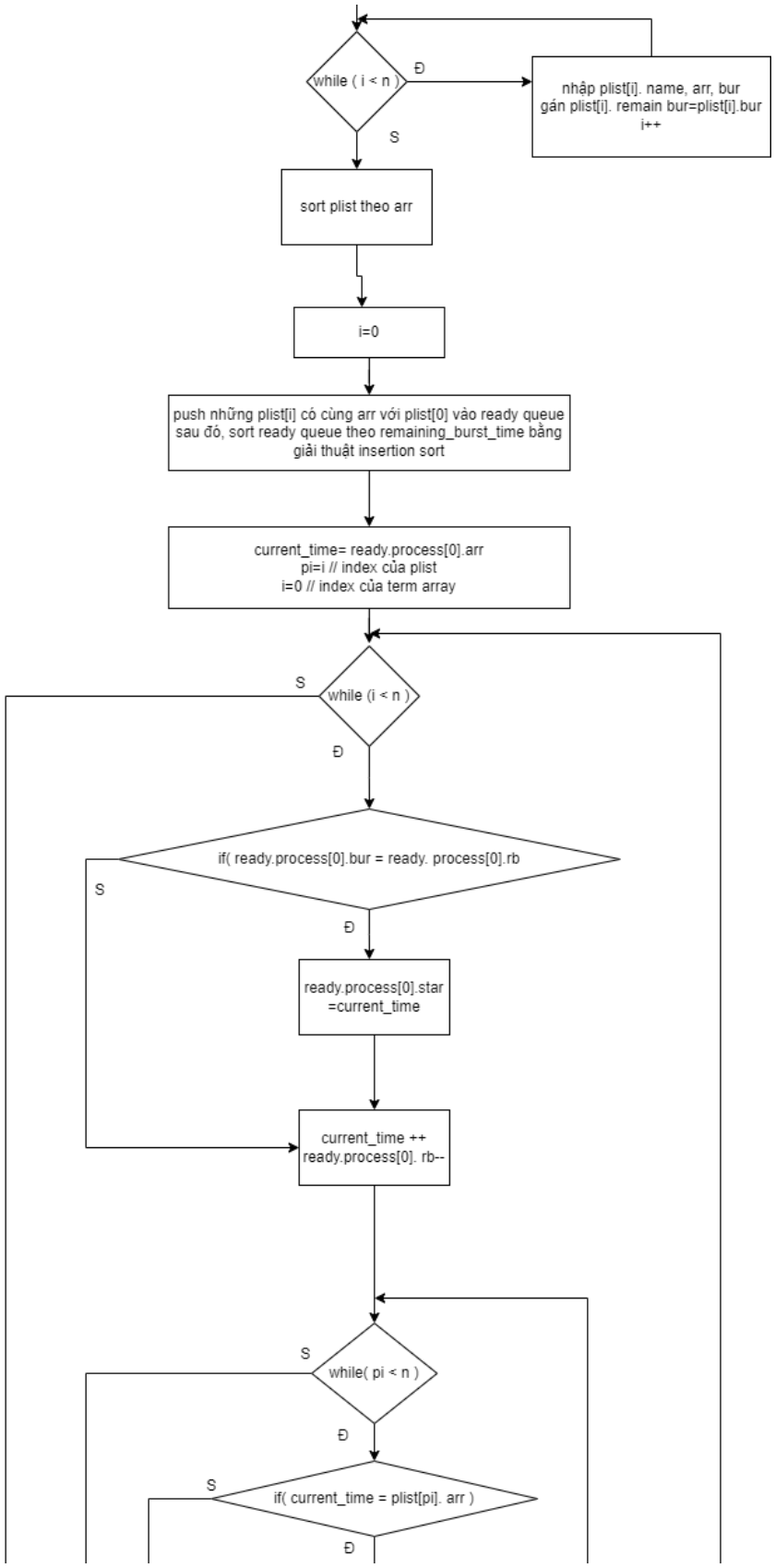
➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

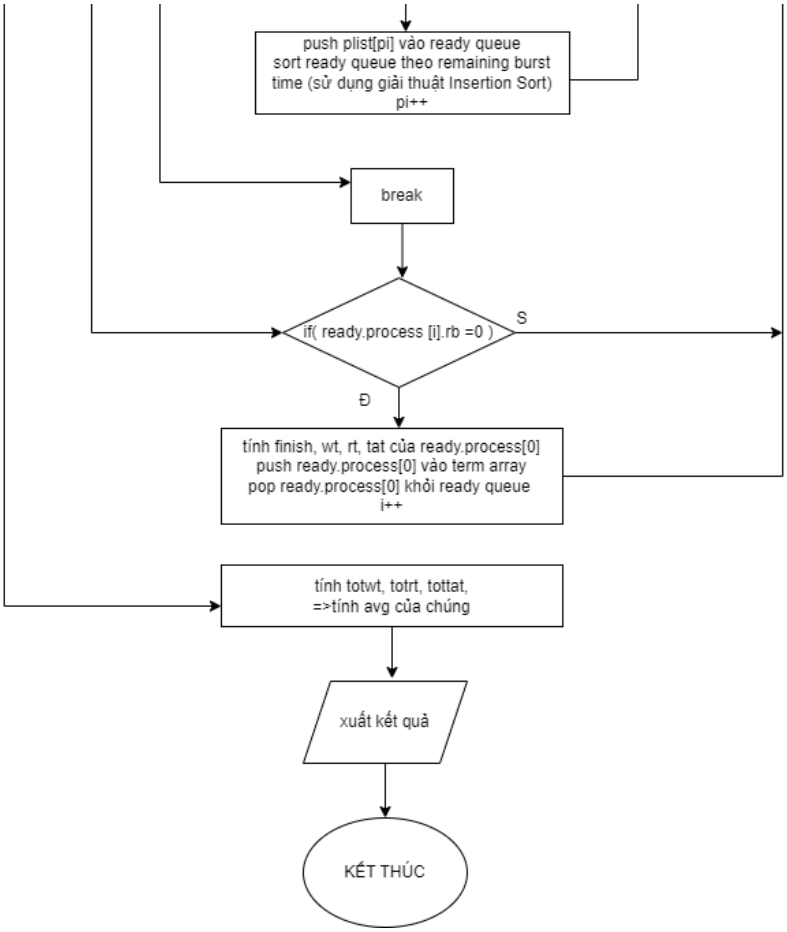
➢ ***Cả 3 test case đều cho kết quả giống với khi giải tay. Vì vậy, ta kết luận code đúng.***

Ex 2. Giải thuật SRTF (Shortest Remaining Time First):

2.1. Vẽ lưu đồ giải thuật




while (i < n)
Đ
nhập plist[i]. name, arr, bur
gán plist[i]. remain bur=plist[i].bur
i++
S
sort plist theo arr
i=0
push những plist[i] có cùng arr với plist[0] vào ready queue sau đó, sort ready queue theo remaining_burst_time bằng giải thuật insertion sort
current_time= ready.process[0].arr
pi=i // index của plist
i=0 // index của term array
S
while (i < n)
Đ
if(ready.process[0].bur = ready. process[0].rb
S
Đ
ready.process[0].star
=current_time
current_time ++
ready.process[0]. rb--
S
while(pi < n)
Đ
S
if(current_time = plist[pi]. arr)
Đ



Hình 19. Lưu đồ giải thuật SRTF

2.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình

Test case: n=5

Process name	Arrival Time	Burst time
1	0	12
2	0	7
3	5	8
4	9	3
5	12	6

Trong minh họa giải thuật dưới đây, một process sẽ được trình bày các thuộc tính lần lượt theo thứ tự [process name, arrival time, burst time, remaining burst time, start, finish, wt, rt, tat]

Thực hiện chạy tay test case trên:

i. Sau quá trình nhập và khởi tạo plist=[[1,0,12,12], [2,0,7,7], [3,5,8,8], [4,9,3,3], [5,12,6,6]], n=5

ii. Ta sử dụng sort theo arr plist=[[1,0,12,12], [2,0,7,7], [3,5,8,8], [4,9,3,3], [5,12,6,6]]
iii. Khởi tạo biến i=0, đồng thời sử dụng vòng lặp để đẩy những process có cùng arr với plist[0] khi này vào hàng đợi ready mỗi lần đẩy vào ready tăng i lên 1 đơn vị cùng với đó sort ready queue theo reaming burst time
iv. ready=[[2,0,7,7], [1,0,12,12]], i=2
v. Gán current_time=0 (=ready[0].bur), pi=2(=i vị trí của plist chưa được push vào ready), gán lại i=0
vi. Sử dụng vòng lặp khi i<5 (0<5)
    a. Tại current_time=0, kiểm tra thấy ready[0].bur=ready[0].rb do đó ready[0].star=0 ⇨ ready=[[2,0,7,7,0], [1,0,12,12]], current_time+=1, ready[0].rb--
    b. Tại current_time=1->3, current_time+=1, ready[0].rb--
    c. Tại current_time=4, current_time+=1, ready[0].rb--, đồng thời kiểm tra thấy current_time==plist[2(=pi))].arr, ta đẩy plist[2] vào trong hàng đợi ready cùng với đó sort ready queue theo rb (sử dụng giải thuật insertion sort), pi++

    ⇨ ready=[[2,0,7,2,0], [3,5,8,8], [1,0,12,12]]

    d. Tại current_time=6, current_time+=1, ready[0].rb-- . Ta kiểm tra thấy ready[0].rb==0. Do đó ready[0] vào hàng đợi terminate đồng thời tính toán finish, wt, rt, tat, pop ready[0] ra khỏi hàng đợi ready, i+=1

    ⇨ terminate =[[2,0,7,0,0,7,0,0,7]], ready=[[3,5,8,8], [1,0,12,12]]

    e. Tại current_time=7, kiểm tra thấy ready[0].bur=ready[0].rb(=8) do đó ready[0].star=7

    ⇨ ready=[[3,5,8,8,7], [1,0,12,12]], currren_time++, ready[0].rb--

    f. Tại current_time=8, current_time++, ready[0].rb-- , đồng thời kiểm tra thấy current_time==plist[3(==pi)].arr, ta đẩy plist[3] vào trong hàng đợi ready cùng với đó sort ready queue theo rb, pi++

    ⇨ ready=[[4,9,3,3], [3,5,8,6,7], [1,0,12,12]]

    g. Tại current_time=9, kiểm tra thấy ready[0].bur=ready[0].rb(=3) do đó ready[0].star=9

    ⇨ ready=[[4,9,3,3,9], [3,5,8,6,7], [1,0,12,12]], current_time++, ready[0].rb--

    h. Tại current_time=11, current_time+=1, ready[0].rb--. Ta kiểm tra thấy current_time==plist[4(==pi)].arr, ta đẩy plist[4] vào trong hàng đợi ready cùng với đó sort ready queue theo rb, pi++. Đồng thời kiểm tra thấy ready[0].rb==0. Do đó đồng thời tính toán finish, wt, rt, tat, push ready[0] vào terminate đồng thời pop ready[0] ra khỏi hàng đợi ready, i+=1

    ⇨ terminate=[[2,0,7,0,0,7,0,0,7], [4,9,3,0,9,12,0,0,3]], ready=[[3,5,8,6,7], [5,12,6,6], [1,0,12,12]]

    i. Tại current_time=17, current_time+=1, ready[0].rb--. Ta kiểm tra thấy có ready[0].rb=0 (ready[0]=[3,5,8,6,7] . Do đó đồng thời tính toán

finish, wt, rt, tat, push ready[0] vào terminate đồng thời pop ready[0] ra khỏi hàng đợi ready, i+=1
⇨ terminate=[[2,0,7,0,0,7,0,0,7], [4,9,3,0,9,12,0,0,3], [3,5,8,0,7,18,5,2,13]], ready=[[5,12,6,6], [1,0,12,12]]

j. Tại current_time=18. Ta kiểm tra thấy ready[0].bur=ready[0].rb(=6) do đó ready[0].star=18
⇨ ready=[[5,12,6,6,18], [1,0,12,12]], current_time+=1, ready[0].rb--

k. Tại current_time=23, current_time+=1, ready[0].rb--. Ta kiểm tra thấy có ready[0].rb=0(ready[0]=[5,12,6,0,18]). Do đó đồng thời tính toán finish, wt, rt, tat, push ready[0] vào terminate đồng thời pop ready[0] ra khỏi hàng đợi ready, i+=1
⇨ terminate=[[2,0,7,0,0,7,0,0,7], [4,9,3,0,9,12,0,0,3], [3,5,8,0,7,18,5,2,13], [5,12,6,0,18,24,6,6,12]], ready=[1,0,12,12]

l. Tại current_time=24. Ta kiểm tra thấy ready[0].bur=ready[0].rb(=12) do đó ready[0].star=24
⇨ ready=[[1,0,12,12,24]], current_time+=1, ready[0].rb--

m. Tại current_time=35, current_time+=1, ready[0].rb--. Ta kiểm tra thấy ready=[[1,0,12,0,24]] thấy có ready[0].rb=0. Do đó đồng thời tính toán finish, wt, rt, tat, push ready[0] vào terminate đồng thời pop ready[0] ra khỏi hàng đợi ready, i+=1,
⇨ terminate=[[2,0,7,0,0,7,0,0,7], [4,9,3,0,9,12,0,0,3], [3,5,8,0,7,18,5,2,13], [5,12,6,0,18,24,6,6,12], [1,0,12,0,24,36,24,24,36]], ready=[]

vii. Ta kiểm tra i==n(5==5) thoát khỏi vòng lặp đồng thời tính avewt=7, avert=6.4, avetat=14.2

viii. Xuất kết quả ra ngoài màn hình

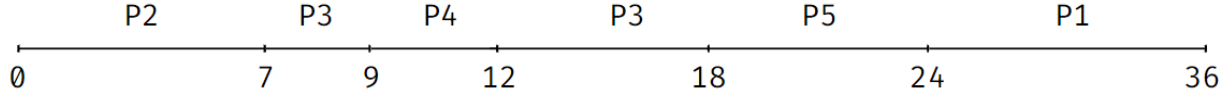
ix. Kết thúc

Kết quả sau khi chạy tay (lưu trong term[]):

Process name	Arrival time	Burst time	Remaining burst	Start	Finish	WT	RT	TAT
2	0	7	0	0	7	0	0	7
4	9	3	0	9	12	0	0	3
3	5	8	0	7	18	5	2	13
5	12	6	0	18	24	6	6	12
1	0	12	0	24	36	24	24	36

avewt=7, avert=6.4, avetat=14.2

So sánh với kết quả giải tay test case trên (thứ tự process name lần lượt từ 1 đến 5):



start[ ]= { 24, 0, 7, 9, 18}

fisnish[ ] ={36, 7, 18, 12, 24}
rt[ ]={ 24, 0, 2, 0, 6 }

⇨ avert= 6.4

wt[ ]={ 24, 0, 5, 0,6 }

⇨ avewt= 7

tat [ ]={ 36,7, 13, 3, 12 }

avert= 14.2

➢ ***Có thể thấy, kết quả thực hiện chạy tay trên lưu đồ giống với kết quả giải tay. Vì vậy, ta kết luận lưu đồ đúng.***

2.3. Thực hiện code cho giải thuật

```c
#include <stdio.h>

/*
Structure P contains the information of a process,
including process name, arrival time, burst time, start time, finish time,
response time, waiting time, turn-around time, and remaining burst time, respec
tively.

The pointers *print and *set point to the printP(used to print the information
of a process)
and setP function(used to calculate its rt, wt and tat).
*/
struct P {
        int pn, arr, bur, star, finish, rt, wt, tat, rb;
        void (*print)(const struct P*);
        void (*set)(struct P*);
};

/*
Structure ReadyQueque contains an array of processes and its size(the number of
 processes in the array).
*/
struct ReadyQueue {
    int size;
    struct P processes[10];
};
```

Hình 20. Tạo cấu trúc Process và Ready Queue trong giải thuật SRTF

- Struct P chứa những thông tin của một process, gồm: tên, arrival time, burst time, start time, finish time, thời gian đáp ứng, thời gian đợi, thời gian hoàn thành và burst time còn lại (remaining burst time). Hai con trỏ *print và *set lần lượt thông qua hàm printP để xuất thông tin của một process ra màn hình, và setP để tính toán RT, WT, TAT.
- Struct ReadyQueue gồm một mảng process và kích cỡ của hàng đợi.

```c
/* The init() function to initialize the ready queue with size = 0.*/
void init(struct ReadyQueue *ready) {
        ready->size=0;
}

/* The setP() function is used to calculate rt, wt and tat of a process p1.*/
void setP(struct P *p1) {
        p1->rt=p1->star-p1->arr;
        p1->tat=p1->finish-p1->arr;
        p1->wt=p1->tat-p1->bur;
}

/* The printP() function is used to print the information of a process p1.*/
void printP(const struct P *p1) {
        printf("%d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\n", p1->pn, p1->arr, p1->
bur, p1->star, p1->finish, p1->rt, p1->wt, p1->tat);
}
```

Hình 21. Hàm khởi tạo Ready Queue, hàm tính RT, WT, TAT và hàm in Process trong giải thuật SRTF

- Hàm init() khởi tạo hàng đợi Ready ban đầu không có phần tử nào.
- Các hàm setP() và printP() thực hiện tính toán và xuất kết quả như hình 21.

```c
/*
The SortArr() function is used to sort a process array plist[] with n elements
in ascending order by arrival time.
*/
void SortArr(struct P plist[], int n) {
        int i, j;
        for(i=0; i<n; i++) {
                for(j=i+1; j<n; j++) {
                        if(plist[i].arr>plist[j].arr) {
                                struct P P0;
                                P0=plist[i];
                                plist[i]=plist[j];
                                plist[j]=P0;
                        }
                }
        }
}
```

Hình 22. Hàm sắp xếp Process theo arrival time tăng dần trong giải thuật SRTF

```c
/*
The SortRB() function is used to sort a process array plist[] with n elements
in ascending order by remaining burst time.

*** This function should use Insertion Sort Algorithm,
so that a new-comming-to-ready-queue process will stand right behind the one
that has the same remaining burst time as it but has arrived before
(in case there're processes having the same remaining burst time,
the one coming first will run first).
*/
void SortRB(struct P plist[], int n) {
        int i, j;
        struct P key;
        for (i = 1; i < n; i++) {
                key = plist[i];
                j = i - 1;
                while (j >= 0 && plist[j].rb > key.rb) {
                        plist[j + 1] = plist[j];
                        j = j - 1;
                }
                plist[j + 1] = key;
        }
}
```

Hình 23. Hàm sắp xếp Process theo remaining burst time tăng dần trong giải thuật SRTF

- Hàm SortRB() sắp xếp một mảng process tăng dần theo remaining burst time. Hàm này sử dụng giải thuật Insertion Sort thay vì sắp xếp thông thường, để khi một process xuất hiện mà có burst time bằng với remaining burst time của process khác đã ở sẵn trong hàng đợi, nó sẽ không tranh mất vị trí của process đến trước mà đứng đợi ngay sau process đến trước đó.

```c
/* The pushP() function pushes a process into the ready queue at the very back
of the queue.*/
void pushP(struct ReadyQueue *ready, struct P p1) {
        ready->processes[ready->size++]=p1;
}

/*
The popP() function pops the first process, which is done running, out of the r
eady queue.
The next process in the queue becomes the first one and soon starts running.
*/
void popP(struct ReadyQueue *ready) {
    for(int i=0; i<ready->size-1; i++)
        ready->processes[i]=ready->processes[i+1];
    ready->size--;
}
```

Hình 24. Hàm thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready trong giải thuật SRTF

- Hàm pushP() và popP() lần lượt thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready, khi đó kích cỡ của hàng đợi cũng sẽ tăng hoặc giảm đi một phần tử tương ứng.

```
int main() {

        /* Define a process array, called plist[]. */
        struct P plist[10];

        /* Define and initialize a ready queue. */
        struct ReadyQueue ready;
        init(&ready);

        /* Define a terminated array, which will contains all the done running
processes. */
        struct P term[10];

        int i, n;

        /* Total waiting time, total turn-around time, total response time of t
he processes. */
        int totwt=0, tottat=0, totrt=0;
```

Hình 25. Khởi tạo mảng process input, hàng đợi Ready, mảng terminated và các biến để tính tổng RT, WT, TAT trong giải thuật SRTF

```
        /*** GET THE INPUT ***/
/* Input of a process includes process name, arrival time and burst time. */
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        for(i=0; i<n; i++) {
                printf("Enter the Process Name, Arrival Time & Burst Time: ");
                scanf("%d%d%d", &plist[i].pn, &plist[i].arr, &plist[i].bur);
                plist[i].print=printP;
                plist[i].set=setP;
                /* rb has inital value the same as burst time value. */
                plist[i].rb=plist[i].bur;
        }

        /* Sort the input in ascending order by arrival time. */
        SortArr(plist, n);
        /* plist[] now contains the processes with increasing arrival time
        (the same arrival times are not sorted). */
```

Hình 26. Nhập input, khởi tạo giá trị ban đầu cho rb và sắp xếp mảng process input theo arrival time

- Sau khi khởi tạo các mảng và hàng đợi, ta tiến hành nhập input gồm số lượng process n, tên, arrival time và burst time của từng process. Ta khởi tạo giá trị ban đầu cho rb (remaining burst time) bằng burst time của chính process đó.
- Các process nhập vào sẽ được lưu trong mảng plist, ta sắp xếp mảng này tăng dần theo arrival time.

```
/* Push the process which arrives earliest into the ready queue.
Use the while loop to push all the processes having the same smallest arrival t
ime into ready queue and sort them by (remaining) burst time. */
        i=0;
        while(plist[i].arr==plist[0].arr) {
                pushP(&ready, plist[i]);
                SortRB(ready.processes, ready.size);
                i++;
        }
```

Hình 27. Đưa tất cả những process cùng xuất hiện sớm nhất vào hàng đợi Ready

- Sử dụng vòng lặp while để push tất cả những process có cùng arrival time với process đến sớm nhất vào hàng đợi Ready. Sau đó, ta tiến hành sắp xếp toàn bộ hàng đợi Ready theo remaining burst time.
- Sau khi push xong những process này, i sẽ chỉ mục đến process có arrival time nhỏ thứ 2 trong plist.

```
/* The variable current_time simulates how the ready queue works in real time.
current_time has initial value of arrival time of the first running process. */
    int current_time = ready.processes[0].arr;

/* pi indexes to the processes in plist[] that haven't been pushed to ready que
ue yet. */
    int pi=i;

/* A process when popped out of ready queue will be added to term[] array at th
e index i, starting at 0. */
    i=0;

/* Keep doing these handling works until all the processes are added to term[].
 */
    while(i<n) {

/* If a process runs for the first time, its start time is current_time. */
        if (ready.processes[0].bur == ready.processes[0].rb)
                        ready.processes[0].star = current_time;

/* When a time unit passes, remaining burst time of running process is reduced
by 1. */
        current_time++;
        ready.processes[0].rb--;
```

Hình 28. Khởi tạo giá trị ban đầu cho current_time, pi, i và tính start time cho process đang chạy

- Ta giả định hàng đợi Ready là một mảng các process, và phần tử đầu tiên trong mảng này luôn là process đang chạy.
- Biến current_time chứa giá trị thời gian hiện tại, dùng để mô phỏng cách hàng đợi Ready làm việc trong thời gian thực. Ta khởi tạo giá trị ban đầu của current_time là thời điểm xuất hiện của process đầu tiên được chọn để thực thi (process đến sớm nhất và có burst time nhỏ nhất so với các process đến cùng lúc).

- Biến pi chỉ mục đến các process trong plist chưa được thêm vào hàng đợi Ready (vì chúng chưa xuất hiện). pi bắt đầu từ vị trí i.
- Biến i đặt lại bằng 0, chỉ mục đến các process trong mảng term[] (là mảng chứa các process đã hoàn thành thực thi).
- Ta bắt đầu xử lý định thời CPU, dừng lại khi tất cả các process trong plist đều đã được thêm vào term[].
- Tại thời điểm current_time, nếu process đầu tiên trong hàng đợi có bur bằng rb (tức nó chưa ở trong trạng thái running trước đó), current_time chính là start time của nó.
- current_time tăng thêm 1, cùng lúc burst time của process đang chạy giảm đi 1.

```
/* Push all the processes arriving at current_time to ready queue and sort the
queue by remaining burst time.
Only do this if there's any process in plist[] not been pushed into ready queue
 yet. */
    while(pi<n) {
            if(plist[pi].arr==current_time) {
                pushP(&ready, plist[pi]);
                SortRB(ready.processes, ready.size);
                pi++;
            }
        /* If at current_time, there's no process arriving, get out of the loop
                        to continue counting current_time. */
            else break;
        }

 /* If remaining burst time of the running process equals 0, it finishes at cur
rent_time.
So, we calculate its rt, wt, and tat via set function.
Then add it to term[] at the index i (increase i by 1 after that), and finally
pop it out of the queue. */
        if(ready.processes[0].rb==0) {
            ready.processes[0].finish=current_time;
            ready.processes[0].set(&ready.processes[0]);
            term[i++]=ready.processes[0];
            popP(&ready);
        }

    }
```

Hình 29. Đưa process vừa xuất hiện vào hàng đợi Ready và sắp xếp, thực hiện các thao tác đối với process đang chạy nếu nó hoàn thành thực thi

- Tại thời điểm current_time mới này, ta sử dụng vòng lặp while để thêm tất cả những process đến tại thời điểm này (arrival time == current_time) vào hàng đợi Ready và tiến hành sắp xếp lại hàng đợi theo burst time của chúng. Ta *thực hiện sắp xếp trên toàn bộ hàng đợi, vì process đầu tiên đang thực thi vẫn có thể bị ngắt trong giải thuật SRTF, nếu remaining burst time của nó lớn hơn burst time của một process khác trong hàng đợi*.
- Nếu tại current_time, không có process nào đến, ta thoát khỏi vòng lặp để tiếp tục đếm current_time.

- Cũng tại thời điểm này, nếu process đang chạy có rb == 0 (tức nó đã chạy xong), finish time của nó chính bằng thời điểm hiện tại. Ta thực hiện tính RT, WT, TAT cho nó thông qua hàm set() và thêm nó vào vị trí i của mảng term[]. Sau đó, ta tăng i lên 1 và xóa process đó ra khỏi hàng đợi Ready. Lúc này, process tiếp theo sẽ được đẩy lên vị trị đầu tiên trong hàng đợi và thực thi.

```
/* Print the information of the terminated processes by their finishing order.
Calculate the total numbers. */
        printf("\nPName\tArrtime\tBurtime\tStart\tFinish\tRT\tWT\tTAT\n");
        for(i=0; i<n; i++) {
                term[i].print(&term[i]);
                totwt+=term[i].wt;
                tottat+=term[i].tat;
                totrt+=term[i].rt;
        }

/* Calculate the average numbers and print them. */
        float avewt, avetat, avert;
        avewt=(float) totwt/n;
        avetat=(float) tottat/n;
        avert=(float) totrt/n;
        printf("Average response time: %0.1f\nAverage waiting time: %0.1f\nAver
age turnaround time: %0.1f\n", avert, avewt, avetat);
}
                                                              212,1         Bot
```

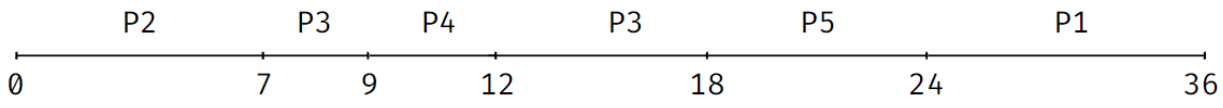
Hình 30. Thực hiện tính toán RT, WT, TAT trung bình và xuất kết quả ra màn hình

2.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình

Test case 1: n=5

Process name	Arrival Time	Burst time
1	0	12
2	0	7
3	5	8
4	9	3
5	12	6

Kết quả giải tay:



Hình 31. Giản đồ Gantt cho test case 1 giải thuật SRTF

start[ ]= { 24, 0, 7, 9, 18}
fisnish[ ] ={36, 7, 18, 12, 24}
rt[ ]={ 24, 0, 2, 0, 6 }
⇨ avert= 6.4
wt[ ]={ 24, 0, 5, 0,6 }
⇨ avewt= 7
tat [ ]={ 36,7, 13, 3, 12 }
⇨ avert= 14.2

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ vim srtf.c
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ gcc srtf.c -o srtf
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./srtf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 12
Enter the Process Name, Arrival Time & Burst Time: 2 0 7
Enter the Process Name, Arrival Time & Burst Time: 3 5 8
Enter the Process Name, Arrival Time & Burst Time: 4 9 3
Enter the Process Name, Arrival Time & Burst Time: 5 12 6

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
2            0       7       0       7       0       0       7
4            9       3       9       12      0       0       3
3            5       8       7       18      2       5       13
5           12       6       18      24      6       6       12
1            0      12       24      36      24      24      36
Average response time: 6.4
Average waiting time: 7.0
Average turnaround time: 14.2
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

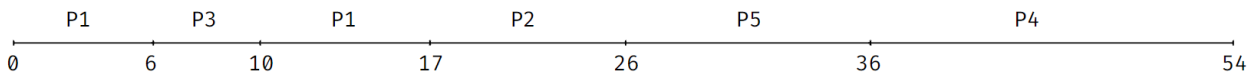
Hình 32. Chạy test case 1 bằng code giải thuật SRTF

➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 2: n=5

Process name	Arrival Time	Burst time
1	0	13
2	4	9
3	6	4
4	7	18
5	12	10

Kết quả giải tay:



Hình 33. Giản đồ Gantt cho test case 2 giải thuật SRTF

start[ ]= { 0, 17, 6, 36, 26 }
fisnish[ ] ={17,26,10, 54, 36}
rt[ ]={ 0, 13,0, 29, 14 }

⇨ avert= 11.2

wt[ ]={ 4, 13,0, 29, 14 }

⇨ avert= 12

tat [ ]={ 17, 22, 4, 47, 24 }

⇨ avert= 22.8

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./srtf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 13
Enter the Process Name, Arrival Time & Burst Time: 2 4 9
Enter the Process Name, Arrival Time & Burst Time: 3 6 4
Enter the Process Name, Arrival Time & Burst Time: 4 7 18
Enter the Process Name, Arrival Time & Burst Time: 5 12 10

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
3            6       4       6       10      0       0       4
1            0       13      0       17      0       4       17
2            4       9       17      26      13      13      22
5           12       10      26      36      14      14      24
4            7       18      36      54      29      29      47
Average response time: 11.2
Average waiting time: 12.0
Average turnaround time: 22.8
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

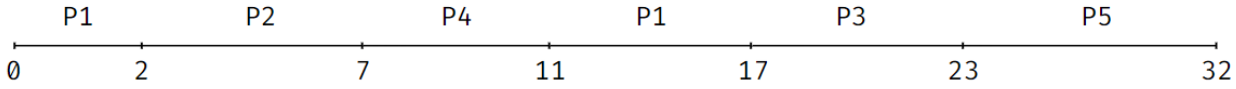
Hình 34. Chạy test case 2 bằng code giải thuật SRTF

➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 3: n=5

Process name	Arrival Time	Burst time
1	0	8
2	2	5
3	3	6
4	5	4
5	7	9

Kết quả giải tay:



Hình 35. Giản đồ Gantt cho test case 3 giải thuật SRTF

start[ ]= {0,2,17,7,23}
finish[ ]= {17,7,23,11,32}
rt[ ]= {0,0,14,2,16}
⇨ avert=6.4
wt[ ]= {9,0,14,2,16}
⇨ averwt=8.2
tat[ ]= {17,5,20,6,25}
⇨ avertat=14.6

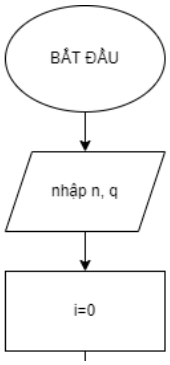
```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./srtf
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 8
Enter the Process Name, Arrival Time & Burst Time: 2 2 5
Enter the Process Name, Arrival Time & Burst Time: 3 3 6
Enter the Process Name, Arrival Time & Burst Time: 4 5 4
Enter the Process Name, Arrival Time & Burst Time: 5 7 9

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
2            2       5       2       7       0       0       5
4            5       4       7       11      2       2       6
1            0       8       0       17      0       9       17
3            3       6       17      23      14      14      20
5            7       9       23      32      16      16      25
Average response time: 6.4
Average waiting time: 8.2
Average turnaround time: 14.6
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

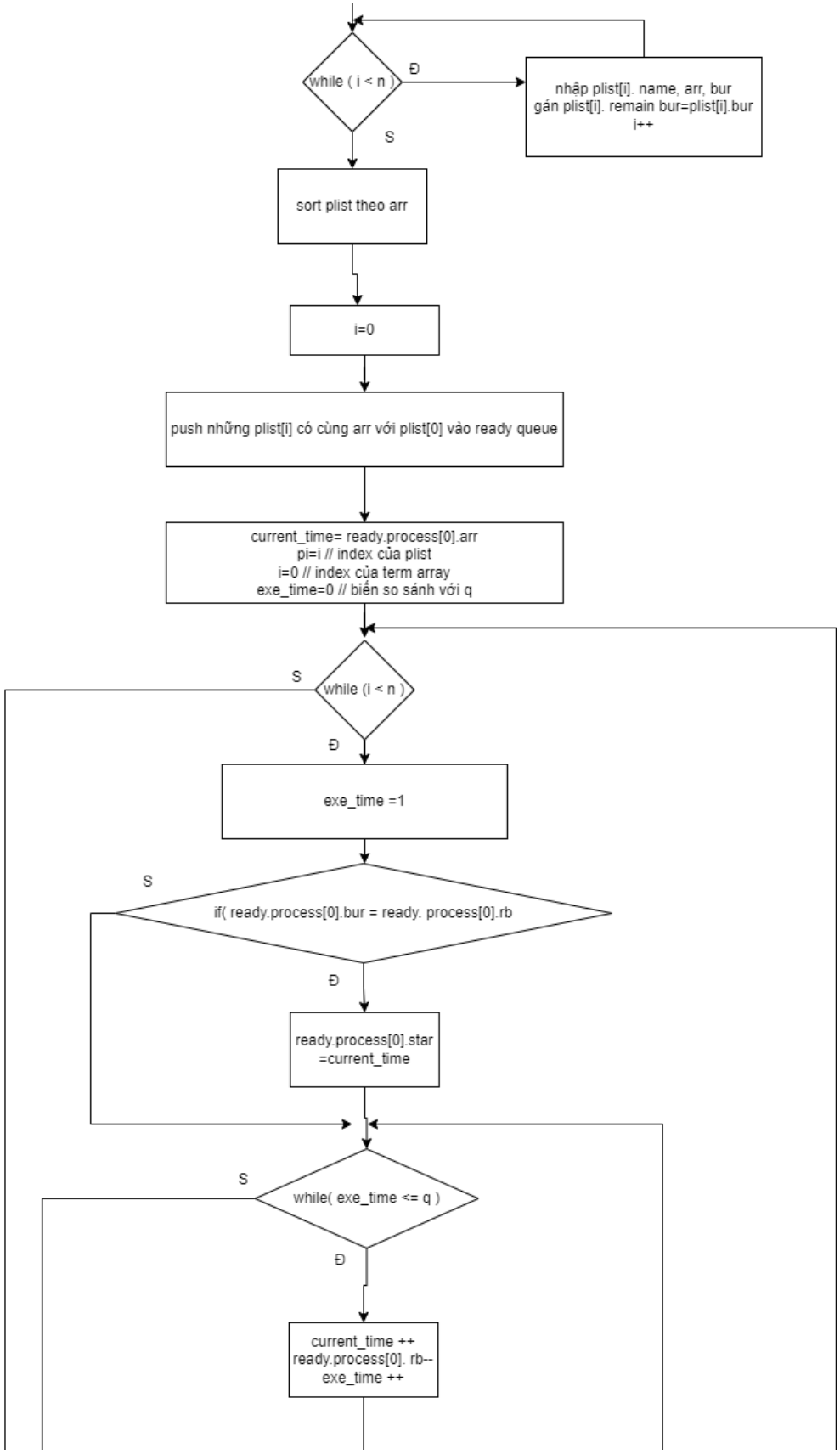
Hình 36. Chạy test case 3 bằng code giải thuật SRTF

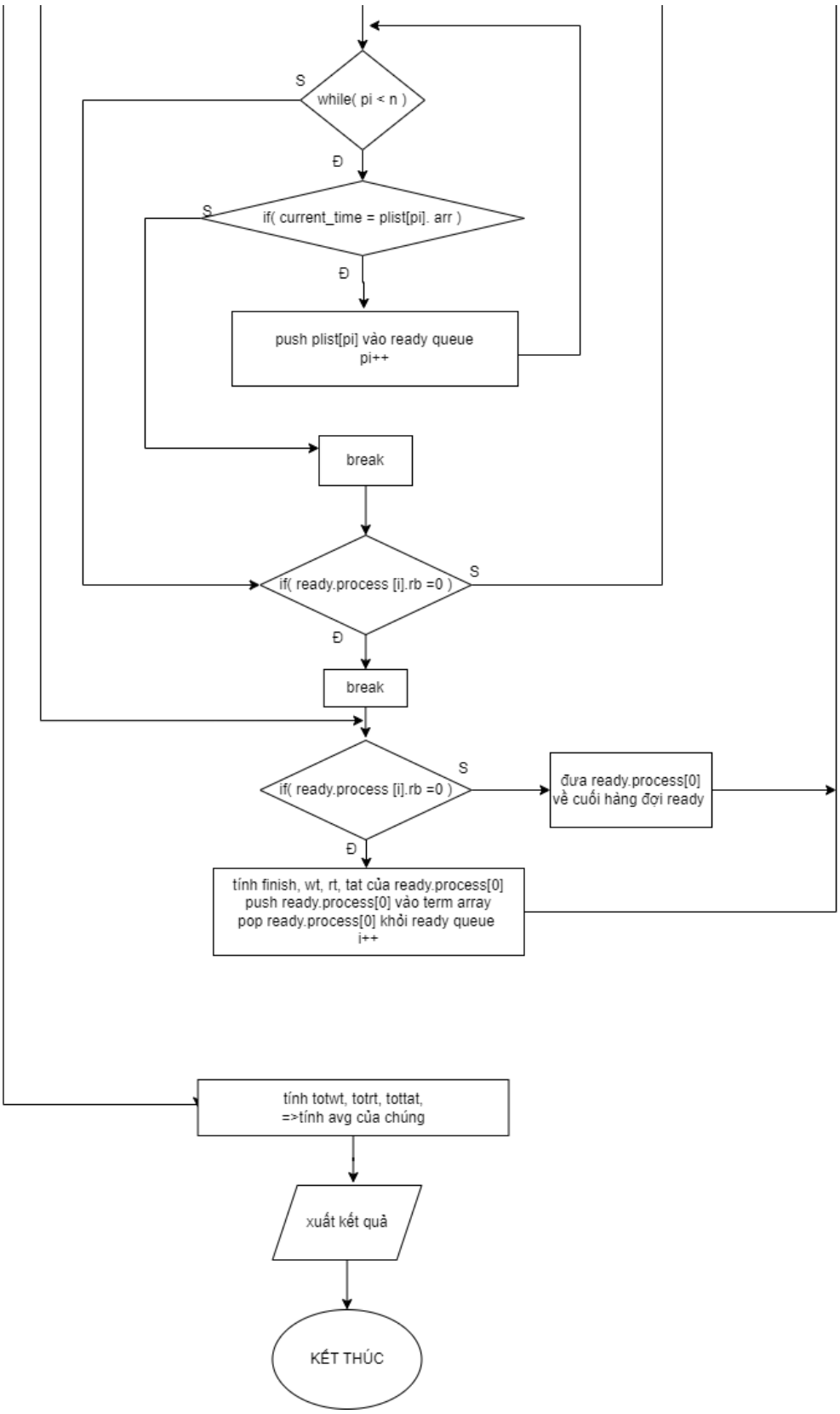
➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***
➢ ***Cả 3 test case đều cho kết quả giống với khi giải tay. Vì vậy, ta kết luận code đúng.***

Ex 3. Giải thuật RR (Round Robin):

3.1. Vẽ lưu đồ giải thuật




while (i < n)
Đ
nhập plist[i]. name, arr, bur
gán plist[i]. remain bur=plist[i].bur
i++
S
sort plist theo arr
i=0
push những plist[i] có cùng arr với plist[0] vào ready queue
current_time= ready.process[0].arr
pi=i // index của plist
i=0 // index của term array
exe_time=0 // biến so sánh với q
S
while (i < n)
Đ
exe_time =1
S
if(ready.process[0].bur = ready. process[0].rb
Đ
ready.process[0].star
=current_time
S
while(exe_time <= q)
Đ
current_time ++
ready.process[0]. rb--
exe_time ++


while(pi < n)
S
Đ
if(current_time = plist[pi]. arr)
S
Đ
push plist[pi] vào ready queue
pi++
break
if(ready.process [i].rb =0)
S
Đ
break
if(ready.process [i].rb =0)
S
đưa ready.process[0]
về cuối hàng đợi ready
Đ
tính finish, wt, rt, tat của ready.process[0]
push ready.process[0] vào term array
pop ready.process[0] khỏi ready queue
i++
tính totwt, totrt, tottat,
=>tính avg của chúng
xuất kết quả
KẾT THÚC

Hình 37. Lưu đồ giải thuật RR

3.2. Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình

Test case: n=5

Process name	Arrival Time	Burst time
1	0	11
2	6	4
3	2	8
4	5	16
5	11	6

Với q=7

Trong minh họa giải thuật dưới đây, một process sẽ được trình bày các thuộc tính lần lượt theo thứ tự [process name, arrival time, burst time, remaining burst time, start, finish, wt, rt, tat]

Thực hiện chạy tay test case trên:

i. Sau quá trình nhập và khởi tạo: plist=[[1,0,11,11], [2,6,4,4], [3,2,8,8], [4,5,16,16], [5,11,6,6]], n=5, q=7
ii. Ta sử dụng sort theo arr plist=[[1,0,11,11], [3,2,8,8], [4,5,16,16], [2,6,4,4], [5,11,6,6]]
iii. Khởi tạo biến i=0, đồng thời sử dụng vòng lặp để đẩy những plist[i] có cùng arr với plist[0] vào hàng đợi ready, đồng thời i+=1 ⇨ ready=[[1,0,11,11]], i=1
iv. Khởi tạo current_time=0 (=ready[0].arr), i=0, pi=1 (i=1), exe_time=0
v. Sử dụng vòng lặp khi i<n (i<5):
    a. Tại current_time =0, gán exe_time=1, kiểm tra thấy ready[0].bur=ready[0].rb (11=11) ⇨ ready[0].start=0 (=current_time)
    b. Thực hiện vòng lặp trong khi exe_time<=q (1<=7) :
        - Khi exe_time=1, current_time++, ready[0].rb--, exe_time++(*)
        - Khi exe_time=2, current_time++, ready[0].rb--, exe_time++. Lúc này current_time= 2 kiểm tra điều kiện current_time=plist[pi(1)].arr (2==2) ta đẩy plist[1] vào hàng đợi ready và pi++
            ⇨ ready=[[1,0,11,9,0], [3,2,8,8]]
        - Từ exe_time=3 -> 4 thực hiện tương tự (*)
        - Khi exe_time=5, current_time++, ready[0].rb--, exe_time++. Lúc này current_time=5 kiểm tra điều kiện current_time=plist[pi(2)].arr (5==5) ta đẩy plist[2] vào hàng đợi ready và pi++ ⇨ ready=[[1,0,11,6,0], [3,2,8,8], [4,5,16,16]]
        - Khi exe_time=6, current_time++, ready[0].rb--, exe_time++. Lúc này current_time=6, kiểm tra điều kiện current_time=plist[pi(3)].arr đẩy vào hàng đợi ready và pi++
            ⇨ ready=[[1,0,11,5,0], [3,2,8,8], [4,5,16,16], [2,6,4,4]]

- Khi exe_time=7, thực hiện tương tự (*)
- Khi exe_time=8, ta thoát khỏi vòng lặp do kiểm tra thấy điều kiện vòng lặp sai, cùng với đó ta đưa ready[0] về cuối hàng đợi ready
    - ⇨ ready =[[3,2,8,8], [4,5,16,16], [2,6,4,4], [1,0,11,4,0]]

c. Tại current_time =7, gán exe_time=1, kiểm tra thấy ready[0].bur=ready[0].rb (8=8) ⇨ ready[0].start=7 (=current_time)
d. Thực hiện vòng lặp trong khi exe_time<=q (1<=7) :
    - Khi exe_time=1->3, current_time++, ready[0].rb--, exe_time++.
    - Khi exe_time=4 , current_time++, ready[0].rb--, exe_time++. Lúc này current_time=11 kiểm tra điều kiện current_time=plist[pi(4)] đẩy vào hàng đợi ready và pi++(pi=5)
        - ⇨ ready=[[3,2,8,4,7], [4,5,16,16], [2,6,4,4], [1,0,11,4,0], [5,11,6,6]], không cần kiểm tra điều kiện để push plist vào ready nữa (vì pi=n)
    - Khi exe_time=5->7, current_time++, ready[0].rb--, exe_time++.
    - Khi exe_time=8, ta thoát khỏi vòng lặp, cùng với đó đưa ready[0] về cuối hàng đợi
        - ⇨ ready=[[4,5,16,16], [2,6,4,4], [1,0,11,4,0], [5,11,6,6], [3,2,8,1,7]]
e. Tại current_time=14, gán exe_time=1 kiểm tra thấy ready[0].bur=ready[0].rb (16=16)
        - ⇨ ready[0].start=14 (=current_time) .
f. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
    - Khi exe_time=1->7 current_time++, ready[0].rb--, exe_time++.
    - Khi exe_time=8, ta thoát khỏi vòng lặp, cùng với đó đưa ready[0] về cuối hàng đợi
        - ⇨ ready=[[2,6,4,4], [1,0,11,4,0], [5,11,6,6], [3,2,8,1,7], [4,5,16,9,14]]
g. Tại current_time=21, gán exe_time=1 kiểm tra thấy ready[0].bur=ready[0].rb(4=4)
        - ⇨ ready[0].start=21 (=current_time)
h. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
    - Khi exe_time=1->3 current_time++, ready[0].rb--, exe_time++.
    - Khi exe_time=4 current_time++, ready[0].rb--, exe_time++. Khi này ready[0].rb=0 vì vậy ta thoát khỏi vòng lặp đồng thời tính finish, wt, rt, tat của ready[0] đồng thời pop ready[0] ra khỏi hàng đợi đưa nó vào hàng đợi terminate, i+=1
        - ⇨ terminate=[[2,6,4,0,21,25,15,15,19]], ready=[[1,0,11,4,0], [5,11,6,6], [3,2,8,1,7], [4,5,16,9,14]]
i. Tại current_time=25, gán exe_time=1
j. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
    - Khi exe_time=1->3 current_time++, ready[0].rb--, exe_time++
    - Khi exe_time=4 current_time++, ready[0].rb--, exe_time++. Khi này ready[0].rb=0 vì vậy ta thoát khỏi vòng lặp đồng thời tính

finish, wt, rt, tat của ready[0] đồng thời pop ready[0] ra khỏi hàng đợi đưa nó vào hàng đợi terminate, i+=1
⇨ terminate=[[2,6,4,0,21,25,15,15,19], [1,0,11,4,0,29,18,0,29]], ready=[[5,11,6,6], [3,2,8,1,7], [4,5,16,9,14]]

- Tại current_time=29, gán exe_time=1 kiểm tra thấy ready[0].bur=ready[0].rb(6=6)
  ⇨ ready[0].start=29 (=current_time)

k. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
l. Khi exe_time=1->5 current_time++, ready[0].rb--, exe_time++
   - Khi exe_time=6 current_time++, ready[0].rb--, exe_time++. Khi này ready[0].rb=0 vì vậy ta thoát khỏi vòng lặp đồng thời tính finish, wt, rt, tat của ready[0] đồng thời pop ready[0] ra khỏi hàng đợi đưa nó vào hàng đợi terminate, i+=1
     ⇨ terminate=[[2,6,4,0,21,25,15,15,19], [1,0,11,4,0,29,18,0,29], [5,11,6,0,29,35,18,18,24]], ready=[[3,2,8,1,7], [4,5,16,9,14]]
m. Tại current_time=35, gán exe_time=1
n. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
   - Khi exe_time=1 current_time++, ready[0].rb--, exe_time++. Khi này ready[0].rb=0 vì vậy ta thoát khỏi vòng lặp đồng thời tính finish, wt, rt, tat của ready[0] đồng thời pop ready[0] ra khỏi hàng đợi đưa nó vào hàng đợi terminate, i+=1
     ⇨ terminate=[[2,6,4,0,21,25,15,15,19], [1,0,11,4,0,29,18,0,29], [5,11,6,0,29,35,18,18,24], [3,2,8,0,7,36,26,5,34]], ready=[[4,5,16,9,14]]
o. Tại current_time=36, gán exe_time=1
p. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
   - Khi exe_time=1->7 current_time++, ready[0].rb--, exe_time++.
   - Khi exe_time=8, ta thoát khỏi vòng lặp do kiểm tra thấy điều kiện vòng lặp sai, cùng với đó ta đưa ready[0] về cuối hàng đợi ready
     ⇨ ready=[[4,5,16,3,14]]
q. Tại current_time=42, gán exe_time=1
r. Thực hiện vòng lặp trong khi exe_time<=q (1<=7):
   - Khi exe_time=1 ->2 current_time++, ready[0].rb--, exe_time++.
   - Khi exe_time=3 Khi này ready[0].rb=0 vì vậy ta thoát khỏi vòng lặp đồng thời tính finish, wt, rt, tat của ready[0] đồng thời pop ready[0] ra khỏi hàng đợi đưa nó vào hàng đợi terminate, i+=1
     ⇨ terminate=[[2,6,4,0,21,25,15,15,19], [1,0,11,4,0,29,18,0,29], [5,11,6,0,29,35,18,18,24], [3,2,8,0,7,36,26,5,34], [4,5,16,0,14,45,24,9,40]]

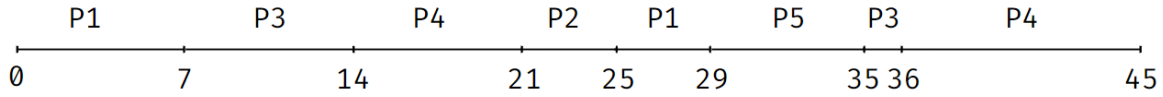
vi. Ta kiểm tra i==n (5==5) thoát khỏi vòng lặp đồng thời tính avewt=20.2, avert=9.4, avetat =29.2
vii. Xuất kết quả ra ngoài màn hình
viii. Kết thúc

Kết quả sau khi chạy tay (lưu trong term[]):

Process name	Arrival time	Burst time	Remaining burst	Start	Finish	WT	RT	TAT
2	6	4	0	21	25	15	15	19
1	0	11	4	0	29	18	0	29
5	11	6	0	29	35	18	18	24
3	2	8	0	7	35	26	5	34
4	5	16	0	14	45	24	9	36

avewt=20.2, avert=9.4, avetat =29.2

So sánh với kết quả giải tay test case trên (thứ tự process name lần lượt từ 1 đến 5):



start[ ]= { 0, 21, 7, 14, 29 }

fisnish[ ] ={29,25,36,45,35}

rt[ ]={ 0, 15, 5, 9, 18 }

⇨ avert= 9.4

wt[ ]={ 18,15,26,24,18 }

⇨ avewt= 20.2

tat [ ]={ 29,19,34,40,24 }

⇨ avetat= 29.2

➢ ***Có thể thấy, kết quả thực hiện chạy tay trên lưu đồ giống với kết quả giải tay. Vì vậy, ta kết luận lưu đồ đúng.***

3.3. Thực hiện code cho giải thuật

```c
#include <stdio.h>

/*
Structure P contains the information of a process,
including process name, arrival time, burst time, start time, finish time,
response time, waiting time, turn-around time, and remaining burst time, respec
tively.

The pointers *print and *set point to the printP(used to print the information
of a process)
and setP function(used to calculate its rt, wt and tat).
*/
struct P {
        int pn, arr, bur, star, finish, rt, wt, tat, rb;
        void (*print)(const struct P*);
        void (*set)(struct P*);
};

/*
Structure ReadyQueque contains an array of processes and its size(the number of
 processes in the array).
*/
struct ReadyQueue {
    int size;
    struct P processes[10];
};
```

Hình 38. Tạo cấu trúc Process và Ready Queue trong giải thuật RR

- Struct P chứa những thông tin của một process, gồm: tên, arrival time, burst time, start time, finish time, thời gian đáp ứng, thời gian đợi, thời gian hoàn thành và burst time còn lại (remaining burst time). Hai con trỏ *print và *set lần lượt thông qua hàm printP để xuất thông tin của một process ra màn hình, và setP để tính toán RT, WT, TAT.
- Struct ReadyQueue gồm một mảng process và kích cỡ của hàng đợi.

```
/* The init() function to initialize the ready queue with size = 0.*/
void init(struct ReadyQueue *ready) {
        ready->size=0;
}

/* The setP() function is used to calculate rt, wt and tat of a process p1.*/
void setP(struct P *p1) {
        p1->rt=p1->star-p1->arr;
        p1->tat=p1->finish-p1->arr;
        p1->wt=p1->tat-p1->bur;
}

/* The printP() function is used to print the information of a process p1.*/
void printP(const struct P *p1) {
        printf("%d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d\n", p1->pn, p1->arr, p1->
bur, p1->star, p1->finish, p1->rt, p1->wt, p1->tat);
}
```

Hình 39. Hàm khởi tạo Ready Queue, hàm tính RT, WT, TAT và hàm in Process trong giải thuật RR

- Hàm init() khởi tạo hàng đợi Ready ban đầu không có phần tử nào.
- Các hàm setP() và printP() thực hiện tính toán và xuất kết quả như hình 39.

```
/*
The SortArr() function is used to sort a process array plist[] with n elements
in ascending order by arrival time.
*/
void SortArr(struct P plist[], int n) {
        int i, j;
        for(i=0; i<n; i++) {
                for(j=i+1; j<n; j++) {
                        if(plist[i].arr>plist[j].arr) {
                                struct P P0;
                                P0=plist[i];
                                plist[i]=plist[j];
                                plist[j]=P0;
                        }
                }
        }
}
```

Hình 40. Hàm sắp xếp Process theo arrival time tăng dần trong giải thuật RR

```c
/* The pushP() function pushes a process into the ready queue at the very back
of the queue.*/
void pushP(struct ReadyQueue *ready, struct P p1) {
        ready->processes[ready->size++]=p1;
}

/*
The popP() function pops the first process, which is done running, out of the r
eady queue.
The next process in the queue becomes the first one and soon starts running.
*/
void popP(struct ReadyQueue *ready) {
    for(int i=0; i<ready->size-1; i++)
        ready->processes[i]=ready->processes[i+1];
    ready->size--;
}
```

Hình 41. Hàm thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready trong giải thuật RR

- Hàm pushP() và popP() lần lượt thêm một process và xóa process đầu tiên ra khỏi hàng đợi Ready, khi đó kích cỡ của hàng đợi cũng sẽ tăng hoặc giảm đi một phần tử tương ứng.

```c
/*
The moveP() function moves the first process, which is running but the quantum
time runs out, to the very back of the ready queue.
The next process in the queue becomes the first one and starts running.
*/
void moveP(struct ReadyQueue *ready) {
    struct P P0;
    P0=ready->processes[0];
    for(int i=0; i<ready->size-1; i++)
        ready->processes[i]=ready->processes[i+1];
    ready->processes[ready->size-1]=P0;
}
```

Hình 42. Hàm di chuyển process đầu tiên về cuối hàng đợi Ready trong giải thuật RR

- Hàm moveP() di chuyển process đầu tiên xuống cuối hàng đợi, do hết quantum time để nhường CPU cho process tiếp theo. Kích cỡ hàng đợi qua hàm này không thay đổi.

```c
int main() {

        /* Define a process array, called plist[]. */
        struct P plist[10];

        /* Define and initialize a ready queue. */
        struct ReadyQueue ready;
        init(&ready);

        /* Define a terminated array, which will contains all the done running
processes. */
        struct P term[10];

        int i, n;

        /* Total waiting time, total turn-around time, total response time of t
he processes. */
        int totwt=0, tottat=0, totrt=0;
```

Hình 43. Khởi tạo mảng process input, hàng đợi Ready, mảng terminated và các biến để tính tổng RT, WT, TAT trong giải thuật RR

```c
    /*** GET THE INPUT ***/
/* Input of a process includes process name, arrival time and burst time. */
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        for(i=0; i<n; i++) {
                printf("Enter the Process Name, Arrival Time & Burst Time: ");
                scanf("%d%d%d", &plist[i].pn, &plist[i].arr, &plist[i].bur);
                plist[i].print=printP;
                plist[i].set=setP;
        /* rb has inital value the same as burst time value. */
                plist[i].rb=plist[i].bur;
        }

    /* Sort the input in ascending order by arrival time. */
        SortArr(plist, n);
    /* plist[] now contains the processes with increasing arrival time. */

    /* Get the quantum time. */
        int q;
        printf("Enter the quantum time: ");
        scanf("%d", &q);
```

Hình 44. Nhập input, khởi tạo giá trị ban đầu cho rb và sắp xếp mảng process input theo arrival time

- Sau khi khởi tạo các mảng và hàng đợi, ta tiến hành nhập input gồm số lượng process n, tên, arrival time và burst time của từng process. Ta khởi tạo giá trị ban đầu cho rb (remaining burst time) bằng burst time của chính process đó.
- Các process nhập vào sẽ được lưu trong mảng plist, ta sắp xếp mảng này tăng dần theo arrival time.
- Khai báo và nhập giá trị cho quantum time q.

```
    /* Push the process which arrives earliest into the ready queue.
       Use the while loop to push all the processes having the same smallest ar
rival time into ready queue. */
        i=0;
        while(plist[i].arr==plist[0].arr) {
                pushP(&ready, plist[i]);
                i++;
    }
```

Hình 45. Đưa tất cả những process cùng xuất hiện sớm nhất vào hàng đợi Ready

- Sử dụng vòng lặp while để push tất cả những process có cùng arrival time với process đến sớm nhất vào hàng đợi Ready.
- Sau khi push xong những process này, i sẽ chỉ mục đến process có arrival time nhỏ thứ 2 trong plist.

```
    /*** THE READY QUEUE IS SUPPOSED TO BE A PROCESS ARRAY
        AND THE INDEX 0 OF THE ARRAY IS ALWAYS THE RUNNING PROCESS. ***/

/* The variable current_time simulates how the ready queue works in real time.
current_time has initial value of arrival time of the first running process. */
    int current_time = ready.processes[0].arr;

/* exe_time calculates execute time of the running process, guaranteeing it won
't exceed quantum time. */
    int exe_time;

/* pi indexes to the processes in plist[] that haven't been pushed to ready que
ue yet. */
    int pi=i;

/* A process when popped out of ready queue will be added to term[] array at th
e index i, starting at 0. */
    i=0;

/* Keep doing these handling works until all the processes are added to term[].
 */
    while (i<n) {

        /* Initialize value of exe_time for each time a process running. */
        exe_time=1;
```

Hình 46. Khởi tạo giá trị ban đầu cho current_time, exe_time, pi, i

- Ta giả định hàng đợi Ready là một mảng các process, và phần tử đầu tiên trong mảng này luôn là process đang chạy.
- Biến current_time chứa giá trị thời gian hiện tại, dùng để mô phỏng cách hàng đợi Ready làm việc trong thời gian thực. Ta khởi tạo giá trị ban đầu của current_time là thời điểm xuất hiện của process đầu tiên được chọn để thực thi (process đến sớm nhất).

- Biến pi chỉ mục đến các process trong plist chưa được thêm vào hàng đợi Ready (vì chúng chưa xuất hiện). pi bắt đầu từ vị trí i.
- Biến i đặt lại bằng 0, chỉ mục đến các process trong mảng term[] (là mảng chứa các process đã hoàn thành thực thi).
- Ta bắt đầu xử lý định thời CPU, dừng lại khi tất cả các process trong plist đều đã được thêm vào term[].
- Biến exe_time đếm thời gian process chạy, đảm bảo mỗi lần sử dụng CPU của nó không vượt quá quantum time. Mỗi khi một process chạy, ta đặt lại giá trị của exe_time=1.

```
    /* If a process runs for the first time, its start time is current_time. */
        if (ready.processes[0].bur == ready.processes[0].rb)
                        ready.processes[0].star = current_time;

   /* Do these until the exe_time of running process exceeds quantum time. */
        while(exe_time<=q) {
/* When a time unit passes, remaining burst time of running process is reduced
by 1 and its exe_time increases by 1. */
            current_time++;
            ready.processes[0].rb--;
            exe_time++;

/* Push all the processes arriving at current_time to ready queue.
 Only do this if there's any process in plist[] not been pushed into ready queu
e yet. */
            while(pi<n) {
                if(plist[pi].arr==current_time) {
                    pushP(&ready, plist[pi]);
                    pi++;
                }
                else break;
            }

/* Get out of the loop when the running process has done running, even if it st
ill has time to execute. */
            if(ready.processes[0].rb==0) break;
        }
```

Hình 47. Tính start time, đếm exe_time và đưa process vừa xuất hiện vào hàng đợi Ready.

- Tại thời điểm current_time, nếu process đầu tiên trong hàng đợi có bur bằng rb (tức nó chưa ở trong trạng thái running trước đó), current_time chính là start time của nó.
- Tiến hành đếm exe_time: current_time tăng thêm 1, cùng lúc burst time của process đang chạy giảm đi 1, và exe_time của nó tăng lên 1.
- Tại thời điểm current_time mới này, ta sử dụng vòng lặp while để thêm tất cả những process đến tại thời điểm này (arrival time == current_time) vào hàng đợi Ready.
- Nếu tại current_time, không có process nào đến, ta thoát khỏi vòng lặp để tiếp tục đếm current_time và exe_time.

- Cũng tại thời điểm này, nếu process đang chạy có rb == 0 (tức nó đã chạy xong), ta thoát khỏi vòng lặp và thực hiện tính toán, dù exe_time chưa vượt quá q.

```
/* If remaining burst time of the running process equals 0, it finishes at curr
ent_time.
So, we calculate its rt, wt, and tat via set function.
Then add it to term[] at the index i (increase i by 1 after that), and finally
pop it out of the queue. */
        if(ready.processes[0].rb==0) {
            ready.processes[0].finish=current_time;
            ready.processes[0].set(&ready.processes[0]);
            term[i++]=ready.processes[0];
            popP(&ready);
        }
/* If it's not yet, move it to the back of the queue and keep waiting for its t
urn to take CPU. */
        else moveP(&ready);
    }
```

Hình 48. Xóa process khỏi hàng đợi hoặc đưa về cuối hàng khi hết quantum time

- finish time của process đang chạy chính bằng thời điểm hiện tại. Ta tính RT, WT, TAT cho nó thông qua hàm set() và thêm nó vào vị trí i của mảng term[]. Sau đó, ta tăng i lên 1 và xóa process đó ra khỏi hàng đợi Ready.
- Nếu process chưa chạy xong nhưng exe_time của nó đã vượt quá q, ta thực hiện di chuyển nó về cuối hàng đợi. Lúc này, process tiếp theo sẽ được đẩy lên vị trị đầu tiên trong hàng đợi và thực thi.

```
/* Print the information of the terminated processes by their finishing order.
Calculate the total numbers. */
        printf("\nPName\tArrtime\tBurtime\tStart\tFinish\tRT\tWT\tTAT\n");
        for(i=0; i<n; i++) {
                term[i].print(&term[i]);
                totwt+=term[i].wt;
                tottat+=term[i].tat;
                totrt+=term[i].rt;
        }

/* Calculate the average numbers and print them. */
        float avewt, avetat, avert;
        avewt=(float) totwt/n;
        avetat=(float) tottat/n;
        avert=(float) totrt/n;
        printf("Average response time: %0.1f\nAverage waiting time: %0.1f\nAver
age turnaround time: %0.1f\n", avert, avewt, avetat);
}
                                                              212,1         Bot
```

Hình 49. Thực hiện tính toán RT, WT, TAT trung bình và xuất kết quả ra màn hình

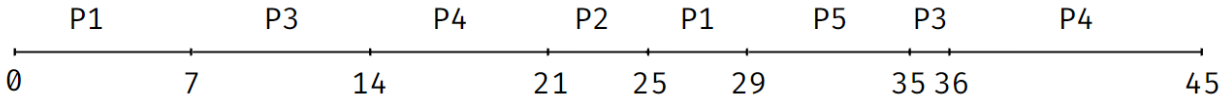
3.4. Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình

Test case 1: n=5

Process name	Arrival Time	Burst time
1	0	11
2	6	4
3	2	8
4	5	16
5	11	6

q=7

Kết quả giải tay:



Hình 50. Giản đồ Gantt cho test case 1 giải thuật RR

start[ ]= { 0, 21, 7, 14, 29 }
fisnish[ ] ={29,25,36,45,35}
rt[ ]={ 0, 15, 5, 9, 18 }
⇨ avert= 9.4
wt[ ]={ 18,15,26,24,18 }
⇨ avewt= 20.2
tat [ ]={ 29,19,34,40,24 }
⇨ avetat= 29.2

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ vim rr.c
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ gcc rr.c -o rr
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./rr
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 11
Enter the Process Name, Arrival Time & Burst Time: 2 6 4
Enter the Process Name, Arrival Time & Burst Time: 3 2 8
Enter the Process Name, Arrival Time & Burst Time: 4 5 16
Enter the Process Name, Arrival Time & Burst Time: 5 11 6
Enter the quantum time: 7

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
2            6       4      21      25      15      15      19
1            0      11       0      29       0      18      29
5           11       6      29      35      18      18      24
3            2       8       7      36       5      26      34
4            5      16      14      45       9      24      40
Average response time: 9.4
Average waiting time: 20.2
Average turnaround time: 29.2
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

Hình 51. Chạy test case 1 bằng code giải thuật RR

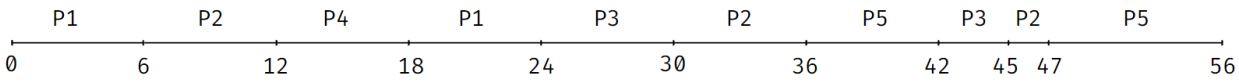
➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 2: n=5

Process name	Arrival Time	Burst time
1	0	12
2	3	14
3	7	9
4	5	6
5	14	15

q=6

Kết quả giải tay:



Hình 52. Giản đồ Gantt cho test case 2 giải thuật RR

start[ ]= {0,6,24,12,36}
fisnish[ ] ={24,47,45,18,56}
rt[ ]={0,3,17,7,22}
⇨ avert= 9.8
wt[ ]={ 12,30,29, 7, 27 }
⇨ avewt= 21
tat [ ]={ 24,44,38,13,42}
⇨ avetat= 32.2

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./rr
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 12
Enter the Process Name, Arrival Time & Burst Time: 2 3 14
Enter the Process Name, Arrival Time & Burst Time: 3 7 9
Enter the Process Name, Arrival Time & Burst Time: 4 5 6
Enter the Process Name, Arrival Time & Burst Time: 5 14 15
Enter the quantum time: 6

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
4            5       6      12      18       7       7      13
1            0      12       0      24       0      12      24
3            7       9      24      45      17      29      38
2            3      14       6      47       3      30      44
5           14      15      36      56      22      27      42
Average response time: 9.8
Average waiting time: 21.0
Average turnaround time: 32.2
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

Hình 53. Chạy test case 2 bằng code giải thuật RR

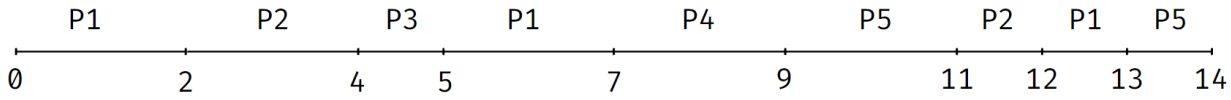
➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

Test case 3: n=5

Process name	Arrival Time	Burst time
1	0	5
2	1	3
3	2	1
4	3	2
5	4	3

q=2

Kết quả giải tay:



Hình 54. Giản đồ Gantt cho test case 3 giải thuật RR

start[ ]= {0,2,4,7,9}
fisnish[ ] ={13,12,5,9, 14}
rt[ ]={0,1,2,4,5}
⇨ avert= 2.4
wt[ ]={8,8,2,4,7}
⇨ avewt=5.8
tat [ ]={13,11,3,6,10}
⇨ avetat= 8.6

```
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$ ./rr
Enter the number of processes: 5
Enter the Process Name, Arrival Time & Burst Time: 1 0 5
Enter the Process Name, Arrival Time & Burst Time: 2 1 3
Enter the Process Name, Arrival Time & Burst Time: 3 2 1
Enter the Process Name, Arrival Time & Burst Time: 4 3 2
Enter the Process Name, Arrival Time & Burst Time: 5 4 3
Enter the quantum time: 2

PName   Arrtime Burtime Start   Finish  RT      WT      TAT
3            2       1       4       5       2       2       3
4            3       2       7       9       4       4       6
2            1       3       2      12       1       8      11
1            0       5       0      13       0       8      13
5            4       3       9      14       5       7      10
Average response time: 2.4
Average waiting time: 5.8
Average turnaround time: 8.6
nguyet-21521211@nguyet21521211-VirtualBox:~/LAB4$
```

Hình 55. Chạy test case 3 bằng code giải thuật RR

➢ ***Kết quả thực hiện bằng code giống với kết quả giải tay.***

➢ ***Cả 3 test case đều cho kết quả giống với khi giải tay. Vì vậy, ta kết luận code đúng.***