

**ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**ĐOCC**



**Tài liệu hướng dẫn thực hành**

**HỆ ĐIỀU HÀNH**

Biên soạn: ThS Phan Đình Duy  
ThS Nguyễn Thanh Thiện  
KS Trần Đại Dương  
KS Trần Hoàng Lộc

---

## MỤC LỤC

|   |           |
|---|-----------|
| <b>BÀI 4. LẬP LỊCH TIẾN TRÌNH .....</b> | <b>1</b>  |
| <b>4.1 Mục tiêu.....</b>                | <b>1</b>  |
| <b>4.2 Nội dung thực hành.....</b>      | <b>1</b>  |
| <b>4.3 Sinh viên chuẩn bị .....</b>     | <b>1</b>  |
| <b>4.4 Hướng dẫn thực hành .....</b>    | <b>9</b>  |
| <b>4.5 Bài tập thực hành .....</b>      | <b>17</b> |
| <b>4.6 Bài tập ôn tập.....</b>          | <b>18</b> |

---




## NỘI QUY THỰC HÀNH

1. Sinh viên tham dự đầy đủ các buổi thực hành theo quy định của giảng viên hướng dẫn (GVHD) (6 buổi với lớp thực hành cách tuần hoặc 10 buổi với lớp thực hành liên tục).
2. Sinh viên phải chuẩn bị các nội dung trong phần “Sinh viên viên chuẩn bị” trước khi đến lớp. GVHD sẽ kiểm tra bài chuẩn bị của sinh viên trong 15 phút đầu của buổi học (nếu không có bài chuẩn bị thì sinh viên bị tính vắng buổi thực hành đó).
3. Sinh viên làm các bài tập ôn tập để được cộng điểm thực hành, bài tập ôn tập sẽ được GVHD kiểm tra khi sinh viên có yêu cầu trong buổi học liền sau bài thực hành đó. Điểm cộng tối đa không quá 2 điểm cho mỗi bài thực hành.


---

## Bài 4. LẬP LỊCH TIẾN TRÌNH

### 4.1 Mục tiêu

-  Sinh viên nắm rõ được các giải thuật: First Come First Served (FCFS), Round Robbin (RR), Shortest Job First (SJF), Shortest Remain Time (SRT).
-  Chỉ ra được ưu điểm và nhược điểm các giải thuật trên.
-  Xây dựng được các chương trình mô phỏng các giải thuật trên.

### 4.2 Nội dung thực hành

-  Mô phỏng giải thuật FCFS

### 4.3 Sinh viên chuẩn bị

Chương trước đã thảo luận về tiến trình – mức trừu tượng của Hệ điều hành đối với mã chương trình đang được thực thi. Chương này thảo luận về lập lịch tiến trình (hay còn gọi là lập lịch CPU) để tìm hiểu cách mà nhân Hệ điều hành cấp phát CPU cho tiến trình thực hiện công việc của nó.

Bộ lập lịch tiến trình sẽ quyết định tiến trình nào chạy, chạy khi nào và chạy trong bao lâu. Bộ lập lịch tiến trình sẽ chia tài nguyên processor time (tài nguyên hữu hạn) giữa các tiến trình đang chạy

---

trong hệ thống. Bằng việc quyết định tiến trình nào sẽ chạy tiếp theo, bộ lập lịch tiến trình có trách nhiệm sử dụng hệ thống tối ưu nhất và phải thể hiện được là nhiều tiến trình đang được thực thi đồng thời.

Hệ điều hành đa nhiệm có thể chạy xen kẽ nhiều tiến trình cùng một lúc. Trên các máy đơn bộ xử lý, các tiến trình sẽ được luân phiên sử dụng bộ xử lý trong một thời gian rất ngắn khiến cho người dùng có cảm giác nhiều tiến trình đang được chạy song song. Trên các máy nhiều bộ xử lý, nhiều tiến trình được thực sự chạy song song cùng nhau trên mỗi bộ xử lý. Dù là đơn bộ xử lý hay đa bộ xử lý, cũng sẽ có các tiến trình tồn tại trong bộ nhớ nhưng bị chặn (không được thực thi) cho đến khi nó được cấp phát processor time để chạy. Trong bài thực hành này, hệ thống đơn bộ xử lý được sử dụng để minh họa bộ lập lịch tiến trình.

Các bản phân phối Linux đều là hệ điều hành đa nhiệm ưu tiên. Trong các hệ điều hành đa nhiệm ưu tiên, bộ lập lịch tiến trình quyết định khi nào một tiến trình ngừng chạy và một tiến trình mới được bắt đầu chạy, việc tạm ngưng một tiến trình đang chạy được gọi là không tiến quyền (tính ưu tiên). Thời gian mà một tiến trình chạy trước khi nó bị chặn là có thể dự đoán được và được gọi là timeslice của tiến trình.

Một tiến trình ở trạng thái đang chạy, nó có thể rời khỏi trạng thái bởi một trong ba lý do sau:

- 
- ✚ Tiến trình đã hoàn thành công việc, khi đó nó trả lại processor time và chuyển sang chờ xử lý kết thúc.
  - ✚ Tiến trình tạm dừng: Khi tiến trình chờ đợi một sự kiện nào đó, tiến trình sẽ được chuyển sang trạng thái thực hiện khi có xuất hiện sự kiện nó đang chờ.
  - ✚ Tiến trình sử dụng hết processor time dành cho nó, khi đó sẽ được chuyển sang trạng thái chờ đến lượt cấp phát tiếp theo.

Việc chuyển tiến trình sang trạng thái chờ về bản chất là thực hiện việc phân phối lại processor time. Để điều khiển tiến trình ở nhiều trạng thái khác nhau, hệ thống thường tổ chức các từ trạng thái (thực chất là các khối điều khiển tiến trình) để ghi nhận tình trạng sử dụng tài nguyên và trạng thái tiến trình. Như vậy, lập lịch tiến trình có nghĩa là tổ chức một hàng đợi các tiến trình sẵn sàng để phân phối processor time cho chúng trên độ ưu tiên của các tiến trình; sao cho hiệu suất sử dụng bộ xử lý là tối ưu nhất. Mỗi tiến trình ở trạng thái sẵn sàng sẽ được gắn với một thứ tự ưu tiên. Thứ tự ưu tiên này được xác định dựa vào các yếu tố như: thời điểm hình thành tiến trình, thời gian thực hiện tiến trình, thời gian kết thúc tiến trình.

---

### 4.3.1 Giải thuật First Come First Served (FCFS)

Trong thuật toán này, độ ưu tiên phục vụ tiến trình căn cứ vào thời điểm hình thành tiến trình. Hàng đợi các tiến trình được tổ chức theo kiểu FIFO (vào trước, ra trước). Mọi tiến trình đều được phục vụ theo trình tự xuất hiện cho đến khi kết thúc hoặc bị ngắt. Ưu điểm của thuật toán này là processor time không bị phân phối lại (không bị ngắt) và chi phí thực hiện thấp nhất (vì không phải thay đổi thứ tự ưu tiên phục vụ, thứ tự ưu tiên là thứ tự của tiến trình trong hàng đợi). Nhược điểm của thuật toán là thời gian trung bình chờ phục vụ của các tiến trình là như nhau (không kể thời gian tiến trình chạy ngắn hay dài), do đó dẫn tới ba điểm sau:

**Ví dụ:**

| Tiến trình | Thời điểm vào | Thời gian thực hiện |
|------------|---------------|---------------------|
| P1         | 0             | 24                  |
| P2         | 1             | 3                   |
| P3         | 2             | 3                   |

Thứ tự cấp phát processor time cho các tiến trình:

| Tiến trình | P1 | P2 | P3 |
|------------|----|----|----|
| Thời điểm  | 0  | 24 | 27 |

Thời gian chờ trung bình:  $(0+23+25)/3=16$ .

---

### 4.3.2 Giải thuật Round robin (RR)

Giải thuật định thời luân phiên (round-robin scheduling algorithm - RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như định thời FCFS nhưng không còn tình trạng độc quyền processor time mà thay vào đó là các tiến trình sẽ lần lượt được cấp phát processor time với một định mức cố định, hay còn gọi là timeslice. Timeslice thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một hàng đợi vòng. Bộ lập lịch tiến trình di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát processor time tới mỗi tiến trình có khoảng thời gian tối đa bằng một timeslice. Để cài đặt định thời RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các tiến trình. Các tiến trình mới được thêm vào đuôi hàng đợi. Bộ lập lịch tiến trình chọn tiến trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 timeslice và gửi tới tiến trình. Sau đó, một trong hai trường hợp sẽ xảy ra. Tiến trình có processor time ít hơn 1 timeslice. Trong trường hợp này, tiến trình sẽ tự giải phóng. Sau đó, bộ lập lịch tiến trình sẽ xử lý tiến trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu processor time của tiến trình đang chạy dài hơn 1 timeslice thì bộ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và tiến trình được đặt trở lại tại đuôi của hàng đợi sẵn sàng. Sau đó, bộ lập lịch tiến trình sẽ chọn tiến trình tiếp theo trong hàng đợi sẵn sàng.





### Ưu điểm:

- ❖ Các tiến trình sẽ được luân phiên cho processor xử lý nên thời gian chờ đợi sẽ ít.
- ❖ Đối với các tiến trình liên quan đến nhập/xuất, người dùng thì rất hiệu quả.
- ❖ Việc cài đặt không quá phức tạp.



### Nhược điểm:

- ❖ Thời gian chờ đợi trung bình theo RR thường là quá dài.
- ❖ Nếu timeslice quá lớn thì RR thành FIFO.
- ❖ Nếu timeslice quá ngắn so với thời gian xử lý của một tiến trình trong danh sách hàng đợi thì việc chờ đợi và xử lý luân phiên sẽ nhiều.
- ❖ Quy tắc là timeslice nên dài hơn 80% processor time.

### Ví dụ:

| Tiến trình | Thời điểm vào | Thời gian thực hiện |
|------------|---------------|---------------------|
| P1         | 0             | 24                  |
| P2         | 1             | 3                   |
| P3         | 2             | 3                   |

timeslice = 4

Thứ tự cấp processor time cho các tiến trình lần lượt là:

| Tiến trình | P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |
|------------|----|----|----|----|----|----|----|----|
| Thời điểm  | 0  | 4  | 7  | 10 | 14 | 18 | 22 | 26 |

Vậy thời gian chờ đợi trung bình sẽ là:  $(3+5+6)/3 = 4,67$ . Như vậy RR có thời gian chờ đợi trung bình nhỏ hơn so với FIFO.

### 4.3.3 Giải thuật Shortest Job First (SJF)

Một tiếp cận khác đối với việc lập lịch tiến trình là giải thuật định thời công việc ngắn nhất trước (shortest job first - SJF). Khi bộ xử lý rảnh, nó được gán tới tiến trình có processor time kế tiếp ngắn nhất. Nếu hai tiến trình có cùng processor time kế tiếp, định thời FCFS sẽ được dùng.



Ưu điểm:

- ❖ Giải thuật được xem là tối ưu, thời gian chờ đợi trung bình giảm.
- ❖ Tận dụng hết năng lực của bộ xử lý.



Nhược điểm:

- ❖ Cài đặt thuật toán phức tạp, tốn nhiều xử lý cho tiến trình quản lý.
- ❖ Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp lập lịch tiến trình ngắn vì không có cách nào để biết chiều dài processor time tiếp theo.
- ❖ Giải thuật SJF có thể độc quyền hay không độc quyền bộ xử lý, dẫn tới giải thuật này có nhiều

---

phiên bản khác nhau và sẽ tối ưu hay không tối ưu phụ thuộc vào chiến lược độc quyền bộ xử lý.

#### 4.3.4 Giải thuật Shortest Remain Time (SRT)

Tương tự như SJF nhưng trong thuật toán này, độ ưu tiên thực hiện các tiến trình dựa vào thời gian cần thiết để thực hiện xong tiến trình (bằng tổng thời gian trừ đi thời gian đã thực hiện). Như vậy, trong thuật toán này cần phải thường xuyên cập nhật thông tin về thời gian đã thực hiện của tiến trình. Đồng thời, chế độ phân bổ lại processor time cũng phải được áp dụng nếu không sẽ làm mất tính ưu việt của thuật toán.



Ưu điểm:

- ❖ Thời gian chờ, tồn tại trong hệ thống của mỗi tiến trình đều ngắn.
- ❖ Các tiến trình ngắn được thực thi nhanh chóng. Hệ thống cần ít chi phí cho việc ra quyết định tiến trình nào sẽ được thực thi tiếp theo.



Nhược điểm:

- ❖ Việc cài đặt thuật toán khá phức tạp.
- ❖ Cần quản lý chặt chẽ việc điều phối các tiến trình.
- ❖ Quản lý thời gian còn lại cho mỗi tiến trình.

#### 4.3.5 Câu hỏi chuẩn bị

1. Vẽ sơ đồ giải thuật của các giải thuật lập lịch tiến trình:

- 
- ❖ FCFS (First Come First Served)
  - ❖ RR (Round Robin)
  - ❖ SJF (Shortest Job First)
  - ❖ SRT (Shortest Remain Time)

2. Giải thích các thuật ngữ sau:

| TT | Thuật ngữ                   | Mô tả |
|----|-----------------------------|-------|
| 1  | Arrival time                |       |
| 2  | Burst time                  |       |
| 3  | Quantum time<br>(timeslice) |       |
| 4  | Response time               |       |
| 5  | Waiting time                |       |
| 6  | Turnaround time             |       |
| 7  | Average waiting time        |       |
| 8  | Average turnaround<br>time  |       |

## 4.4 Hướng dẫn thực hành

### 4.4.1 Phân tích bài toán

Trong phần này, chúng ta sẽ đi hiện thực và mô phỏng giải thuật FCFS. Trước hết, để mô phỏng các giải thuật, ta cần xác định các mục tiêu mô phỏng bao gồm:

- ❖ Vẽ giản đồ Gantt cho các tiến trình
- ❖ Tính toán các thông số Start time, Finish time, Waiting time, Response time, và Turnaround time
- ❖ Tính toán các thông số Average Waiting time và Average Turnaround time

Từ đó, ta xác định INPUT và OUTPUT của bài toán bao gồm:

| Với mỗi tiến trình  |                 |
|---|-----------------|
| INPUT   | OUTPUT          |
| PID   | Start time      |
| Arrival Time  | Finish time     |
| Burst Time  | Waiting time    |
|   | Response time   |
|   | Turnaround time |
| <b>OUTPUT cho toàn bài:</b><br>Giản đồ Gantt<br>Average Waiting time<br>Average Turnaround time |                 |

Áp dụng cách thức quản lý của hệ điều hành, trong chương trình mô phỏng, ta cũng sẽ tổ chức một kiểu dữ liệu tên là PCB để chứa các thông tin cần thiết của tiến trình bao gồm các input và output như trên.

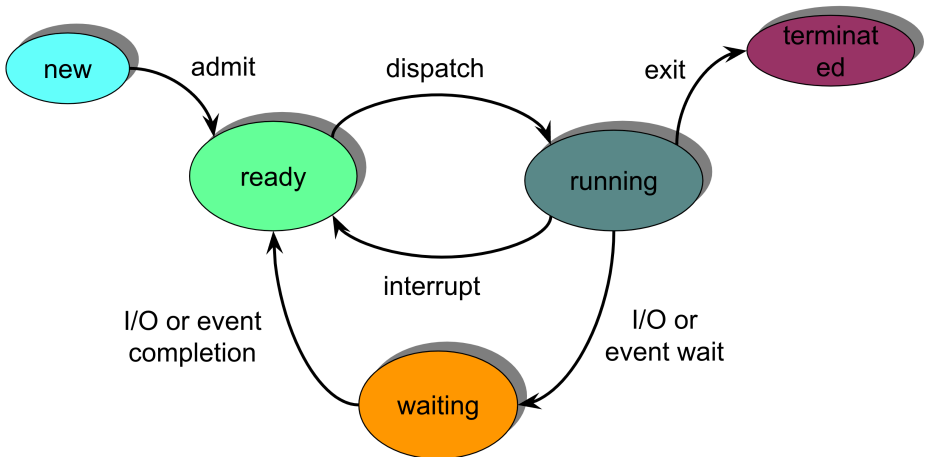
```
typedef struct{
    int iPID;
```

```

int iArrival, iBurst;
int iStart, iFinish, iWaiting, iResponse, iTaT;
} PCB;

```

Dựa trên lý thuyết, ta có sơ đồ chuyển trạng thái của các tiến trình như sau:



Hình 1. Sơ đồ chuyển trạng thái của tiến trình

Trong đó, các giải thuật định thời được sử dụng trong giai đoạn dispatch - chuyển tiến trình từ trạng thái ready sang running. Dưới góc độ lập trình, để mô phỏng tính toán cho các giải thuật định thời, ta có thể tổ chức các trạng thái của tiến trình thành các mảng như sau:

| Input [5] | ReadyQueue [5] | TerminatedArray [5] |
|-----------|----------------|---------------------|
| P1 (0, 6) |                |                     |
| P2 (2, 7) |                |                     |
| P3 (5, 8) |                |                     |
| P4 (9, 3) |                |                     |

|             |            |                 |
|-------------|------------|-----------------|
| P5 (12, 6)  |            |                 |
| iRemain = 5 | iReady = 0 | iTerminated = 0 |

Trong đó:

- ❖ Mảng `Input[]`: lưu các tiến trình do người dùng nhập vào ban đầu. P1 (0, 12) có nghĩa là tiến trình P1 có `iArrival = 0` và `iBurst = 12`.
- ❖ Mảng `ReadyQueue[]`: mô phỏng trạng thái Ready của tiến trình
- ❖ Mảng `TerminatedArray[]`: lưu các tiến trình đã hoàn thành

Giả sử, P1, P2, P3, P4, P5 được sắp xếp theo thứ tự `iArrival`, như vậy, P1 sẽ tiến trình đầu tiên vào `ReadyQueue[]`. Lưu ý, do chúng ta không mô phỏng trạng thái Running của tiến trình, nên có thể đặt giả định tiến trình đứng đầu `ReadyQueue[]`, cụ thể là `ReadyQueue[0]`, sẽ tiến trình được chạy. Việc "chạy" này được mô phỏng bằng cách tính toán các thông số OUTPUT.

| <b>Input[5]</b> | <b>ReadyQueue[5]</b> | <b>TerminatedArray[5]</b> |
|-----------------|----------------------|---------------------------|
| P2 (2, 7)       | P1 (0, 6)            |                           |
| P3 (5, 8)       |                      |                           |
| P4 (9, 3)       |                      |                           |
| P5 (12, 6)      |                      |                           |
|                 |                      |                           |
| iRemain = 4     | iReady = 1           | iTerminated = 0           |

Trong thời gian P1 thực thi, sẽ có các tiến trình khác xuất hiện, ta có thể dựa vào thông số `iArrival` của tiến trình để biết được tiến

trình nào sẽ được thêm vào ReadyQueue[] trong khi P1 được thực thi. Trong bảng bên dưới, ta thấy, trong khi P1 đang "thực thi" từ thời điểm 0 đến 6, P2 và P3 xuất hiện nên sẽ được đưa vào ReadyQueue[].

| Input[5]    | ReadyQueue[5] | TerminatedArray[5] |
|-------------|---------------|--------------------|
| P4 (9, 3)   | P1 (0, 6)     |                    |
| P5 (12, 6)  | P2 (2, 7)     |                    |
|             | P3 (5, 8)     |                    |
|             |               |                    |
|             |               |                    |
| iRemain = 2 | iReady = 3    | iTerminated = 0    |

Sau khi đã thêm đủ các tiến trình vào ReadyQueue[], P1 "thực thi" hoàn tất thì có thể chuyển sang mảng Terminated Array[]. Lúc này ReadyQueue[0] là tiến trình P2 sẽ được thực thi.

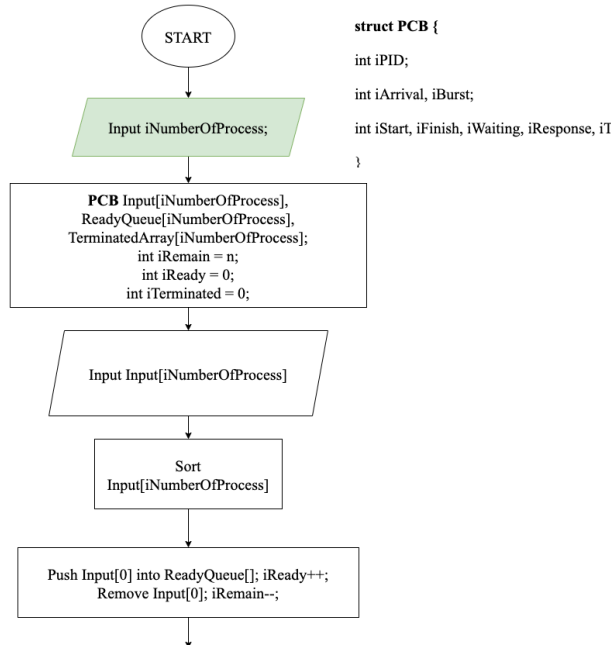
| Input[5]    | ReadyQueue[5] | TerminatedArray[5] |
|-------------|---------------|--------------------|
| P4 (9, 3)   | P2 (2, 7)     | P1 (0, 6)          |
| P5 (12, 6)  | P3 (5, 8)     |                    |
|             |               |                    |
|             |               |                    |
|             |               |                    |
| iRemain = 2 | iReady = 2    | iTerminated = 1    |

Tiếp tục thực thi theo quy tắc trên cho đến khi iTerminated = 5, chính là số lượng tiến trình ban đầu nhập vào thì giải thuật kết thúc.



### 4.4.2 Lưu đồ giải thuật

Trong Hình 2 là phần đầu của lưu đồ giải thuật FCFS thực hiện việc khai báo và nhập dữ liệu.



Hình 2. Phần đầu của lưu đồ giải thuật FCFS

Trao đổi với nhóm của mình và hãy hoàn thiện phần còn lại của lưu đồ giải thuật FCFS, sau đó, trình bày trước lớp.

### 4.4.3 Hiện thực giải thuật FCFS trên C

```
#include <stdio.h>
#include <stdlib.h>

#define SORT_BY_ARRIVAL 0
#define SORT_BY_PID 1
#define SORT_BY_BURST 2
```

```

#define SORT_BY_START 3

typedef struct{
    int iPID;
    int iArrival, iBurst;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
} PCB;

void inputProcess(int n, PCB P[])

void printProcess(int n, PCB P[])

void exportGanttChart (int n, PCB P[])

void pushProcess(int *n, PCB P[], PCB Q)

void removeProcess(int *n, int index, PCB P[])

int swapProcess(PCB *P, PCB *Q)

int partition (PCB P[], int low, int high, int iCriteria)

void quickSort(PCB P[], int low, int high, int iCriteria)

void calculateAWT(int n, PCB P[])

void calculateATaT(int n, PCB P[])

int main()
{
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB TerminatedArray[10];

    int iNumberOfProcess;
    printf("Please input number of Process: ");
    scanf("%d", &iNumberOfProcess);

```

```

    int iRemain = iNumberOfProcess, iReady = 0, iTerminated =
0;

    inputProcess(iNumberOfProcess, Input);
    quickSort(Input, 0, iNumberOfProcess - 1,
SORT_BY_ARRIVAL);

    pushProcess(&iReady, ReadyQueue, Input[0]);
    removeProcess(&iRemain, 0, Input);

    ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
    ReadyQueue[0].iFinish = ReadyQueue[0].iStart +
ReadyQueue[0].iBurst;
    ReadyQueue[0].iResponse = ReadyQueue[0].iStart -
ReadyQueue[0].iArrival;
    ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
    ReadyQueue[0].iTaT = ReadyQueue[0].iFinish -
ReadyQueue[0].iArrival;

    printf("\nReady Queue: ");
    printProcess(1, ReadyQueue);

    while (iTerminated < iNumberOfProcess)
    {
        while (iRemain > 0)
        {
            if (Input[0].iArrival <= ReadyQueue[0].iFinish)
            {
                pushProcess(&iReady, ReadyQueue, Input[0]);
                removeProcess(&iRemain, 0, Input);
                continue;
            }
            else
                break;
        }

        if (iReady > 0)
        {

```

```

        pushProcess(&iTerminated, TerminatedArray,
ReadyQueue[0]);
        removeProcess(&iReady, 0, ReadyQueue);

        ReadyQueue[0].iStart = TerminatedArray[iTerminated
- 1].iFinish;
        ReadyQueue[0].iFinish = ReadyQueue[0].iStart +
ReadyQueue[0].iBurst;
        ReadyQueue[0].iResponse = ReadyQueue[0].iStart -
ReadyQueue[0].iArrival;
        ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
        ReadyQueue[0].iTaT = ReadyQueue[0].iFinish -
ReadyQueue[0].iArrival;
    }
}

printf("\n==== FCFS Scheduling ==== \n");
exportGanttChart(iTerminated, TerminatedArray);

quickSort(TerminatedArray, 0, iTerminated - 1,
SORT_BY_PID);

calculateAWT(iTerminated, TerminatedArray);
calculateATaT(iTerminated, TerminatedArray);

return 0;
}

```




Source code trên thể hiện hàm main của giải thuật FCFS. Hãy hoàn thành các hàm hỗ trợ để có thể thực thi được chương trình trên.

## 4.5 Bài tập thực hành

Cho các yêu cầu sau:



Vẽ lưu đồ giải thuật

- 
-  Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình
  -  Thực hiện code cho giải thuật, trong đó, Arrival Time của mỗi tiến trình được tạo ngẫu nhiên trong đoạn  $[0, 20]$ , Burst Time của mỗi tiến trình được tạo ngẫu nhiên trong đoạn  $[2, 12]$
  -  Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình, so sánh kết quả chạy tay và chạy code

1. Viết chương trình mô phỏng giải thuật SJF với các yêu cầu trên.
2. Chọn một trong hai hai giải thuật gồm SRTF và RR để thực hiện các yêu cầu trên.

## **4.6 Bài tập ôn tập**

1. Thực hiện giải thuật còn lại trong câu 2 phần 4.5.