Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

Họ và tên: Đoàn Phương Nam-Trần Quỳnh Thi-Trương Thiên Lộc-Phạm Duy Long
Mã số sinh viên:22520908-22521461-21520330-20521573
Lớp: IT007.O11.2

# HỆ ĐIỀU HÀNH
# BÁO CÁO LAB 4

**CHECKLIST**

**3.5. BÀI TẬP THỰC HÀNH**

| | BT 1 | BT 2 |
|---|---|---|
| **Vẽ lưu đồ giải thuật** | ☒ | ☒ |
| **Chạy tay lưu đồ giải thuật** | ☒ | ☒ |
| **Hiện thực code** | ☒ | ☒ |
| **Chạy code và kiểm chứng** | ☒ | ☒ |

**3.6. BÀI TẬP ÔN TẬP**

| | BT 1 |
|---|---|
| **Vẽ lưu đồ giải thuật** | ☒ |
| **Chạy tay lưu đồ giải thuật** | ☒ |
| **Hiện thực code** | ☒ |
| **Chạy code và kiểm chứng** | ☒ |

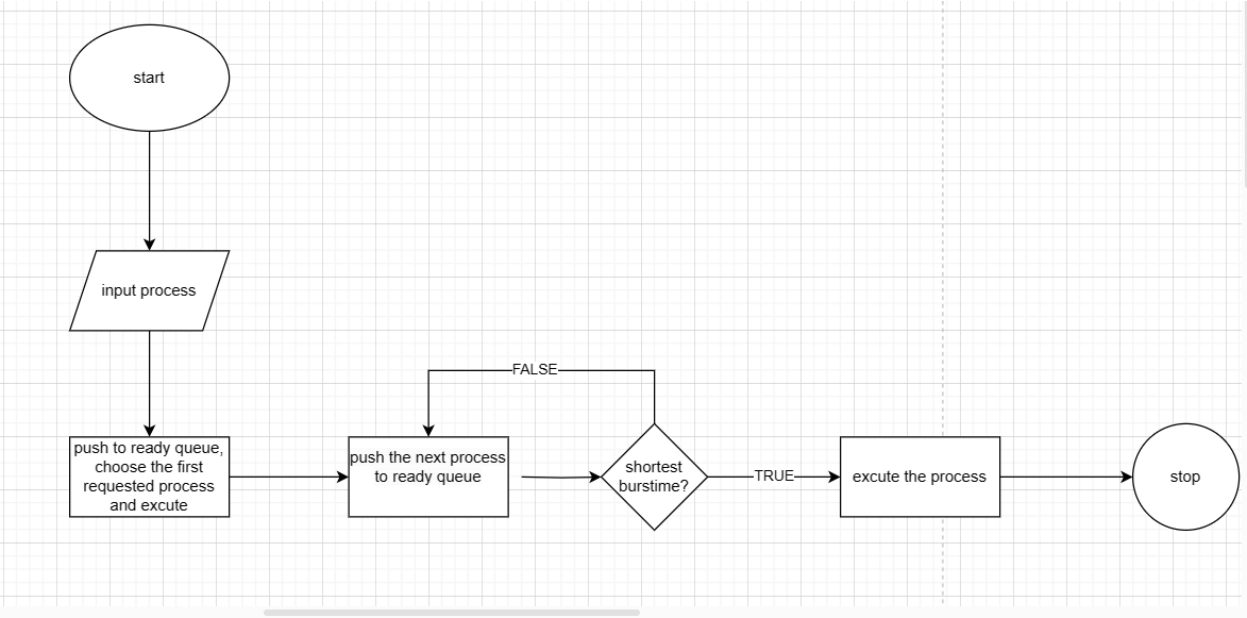
**Tự chấm điểm:** 10/10

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*
***<Tên nhóm>_LAB3.pdf***

## 2.5. BÀI TẬP THỰC HÀNH

### 1. Giải thuật Shortest-Job-First

**Vẽ lưu đồ thuật toán:**




**Chạy tay 1 test case:**

*test case 1:*

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


- Thời gian đợi trung bình: 9.6
- Thời gian hoàn thành trung bình: 16.8

**CODE:**

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

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

void inputProcess(int n, PCB P[]) {
  for (int i = 0; i < n; i++) {
    printf("Input Process %d\n", i + 1);
    printf("PID: ");
    scanf("%d", &P[i].iPID);
    printf("Arrival Time: ");
    scanf("%d", &P[i].iArrival);
    printf("Burst Time: ");
    scanf("%d", &P[i].iBurst);
    P[i].iRemaining = P[i].iBurst;
    P[i].iResponse = -1;
  }
}

void ExportGanttChart(int a[], int totalTime) {
    printf("P%d\t", a[0]);
    for (int iTime = 1; iTime <= totalTime; ++iTime) {
        if (a[iTime] == a[iTime - 1]) continue;
        printf("P%d\t", a[iTime]);
    }
    printf("\n0\t");
    for (int iTime = 1; iTime <= totalTime; ++iTime) {
        if (a[iTime] == a[iTime - 1]) continue;
        printf("%d\t", iTime);
    }
    printf("%d\n", totalTime + 1);
}

void calculateAWT(int n, PCB P[]) {
  float totalWaitingTime = 0;
  float averageWaitingTime = 0;

  for (int i = 0; i < n; i++) {
    P[i].iWaiting = P[i].iStart - P[i].iArrival;
```

```
45.     totalWaitingTime += P[i].iWaiting;
46.   }
47.
48.   if (n > 0) {
49.     averageWaitingTime = totalWaitingTime / n;
50.   }
51.
52.   printf("Average Waiting Time: %.2f\n", averageWaitingTime);
53. }
54.
55. void calculateATaT(int n, PCB P[]) {
56.   float totalTurnaroundTime = 0;
57.   float averageTurnaroundTime = 0;
58.
59.   for (int i = 0; i < n; i++) {
60.     P[i].iTaT = P[i].iFinish - P[i].iArrival;
61.     totalTurnaroundTime += P[i].iTaT;
62.   }
63.
64.   if (n > 0) {
65.     averageTurnaroundTime = totalTurnaroundTime / n;
66.   }
67.
68.   printf("Average Turnaround Time: %.2f\n", averageTurnaroundTime);
69. }
70.
71. void swapProcess(PCB *P, PCB *Q) {
72.   PCB temp = *P;
73.   *P = *Q;
74.   *Q = temp;
75. }
76.
77. int partition(PCB P[], int low, int high) {
78.   int pivot = P[high].iBurst;
79.   int i = low - 1;
80.
81.   for (int j = low; j <= high - 1; j++) {
82.     if (P[j].iBurst < pivot) {
83.       i++;
84.       swapProcess(&P[i], &P[j]);
85.     }
86.   }
87.
88.   swapProcess(&P[i + 1], &P[high]);
89.   return (i + 1);
90. }
```

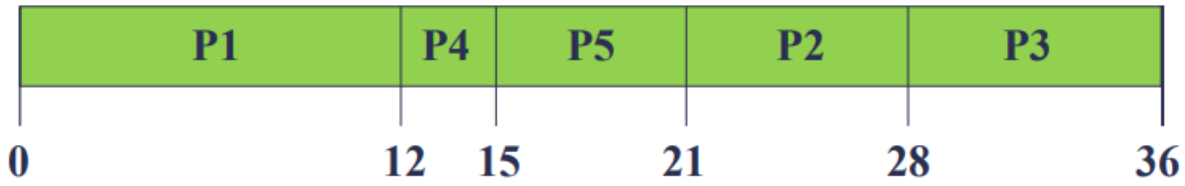
```
91.
92. void quickSort(PCB P[], int low, int high) {
93.   if (low < high) {
94.     int pi = partition(P, low, high);
95.
96.     quickSort(P, low, pi - 1);
97.     quickSort(P, pi + 1, high);
98.   }
99. }
100.
101.  int main() {
102.    int iNumberOfProcess;
103.
104.    printf("Please input number of Processes: ");
105.    scanf("%d", &iNumberOfProcess);
106.
107.    PCB Input[iNumberOfProcess];
108.
109.    inputProcess(iNumberOfProcess, Input);
110.    quickSort(Input, 0, iNumberOfProcess - 1);
111.
112.    int currentTime = 0;
113.    int remainingProcesses = iNumberOfProcess;
114.    int selectedProcess = -1;
115.    int minBurst = 999999; // Giá trị burst time nhỏ nhất hiện tại
116.
117.    while (remainingProcesses > 0) {
118.      for (int i = 0; i < iNumberOfProcess; i++) {
119.        if (Input[i].iArrival <= currentTime && Input[i].iRemaining > 0) {
120.          if (Input[i].iBurst < minBurst) {
121.            minBurst = Input[i].iBurst;
122.            selectedProcess = i;
123.          }
124.        }
125.      }
126.
127.      if (selectedProcess == -1) {
128.        currentTime++;
129.      } else {
130.        Input[selectedProcess].iStart = currentTime;
131.        Input[selectedProcess].iFinish =
132.            Input[selectedProcess].iStart + Input[selectedProcess].iRemaining;
133.        currentTime = Input[selectedProcess].iFinish;
134.        Input[selectedProcess].iRemaining = 0;
135.        remainingProcesses--;
136.        minBurst = 999999;
```

```
137.       selectedProcess = -1;
138.     }
139.   }
140.
141.   printf("\n===== SJF Scheduling =====\n");
142.   calculateAWT(iNumberOfProcess, Input);
143.   calculateATaT(iNumberOfProcess, Input);
144.
145.   // Tạo và hiển thị biểu đồ Gantt
146.   int ganttChart[currentTime];
147.   for (int i = 0; i < iNumberOfProcess; ++i) {
148.     for (int j = Input[i].iStart; j < Input[i].iFinish; ++j) {
149.       ganttChart[j] = Input[i].iPID;
150.     }
151.   }
152.
153.   ExportGanttChart(ganttChart, currentTime - 1);
154.
155.   return 0;
156. }
```

**Test case 1:**

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

| | | |
|---|---|---|
| P5 | 12 | 6 |




- Thời gian đợi trung bình: 9.6
- Thời gian hoàn thành trung bình: 16.8

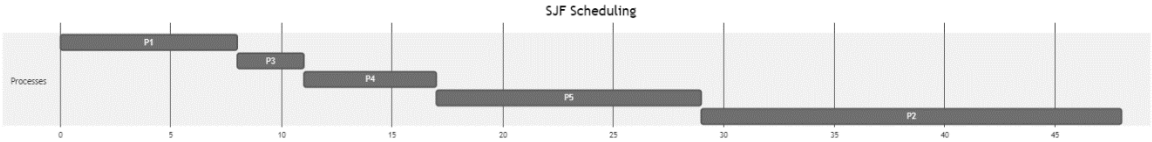
```
Please input number of Processes: 5
Input Process 1
PID: 1
Arrival Time: 0
Burst Time: 12
Input Process 2
PID: 2
Arrival Time: 2
Burst Time: 7
Input Process 3
PID: 3
Arrival Time: 5
Burst Time: 8
Input Process 4
PID: 4
Arrival Time: 9
Burst Time: 3
Input Process 5
PID: 5
Arrival Time: 12
Burst Time: 6

===== SJF Scheduling =====
Average Waiting Time: 9.60
Average Turnaround Time: 16.80
P1  P4  P5  P2  P3
0   12  15  21  28  36
```

**Test case 2:**

| Process | Arrival Time | CPU Burst Time |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 2 | 19 |
| P3 | 4 | 3 |

| | | |
|---|---|---|
| P4 | 5 | 6 |
| P5 | 7 | 12 |

**1. SJF**




- Thời gian đợi trung bình: (0 + 4 + 6 + 10 + 27) / 5 = 9.4
- Thời gian hoàn thành trung bình: (8 + 7 + 12 + 22 + 46) / 5 = 19

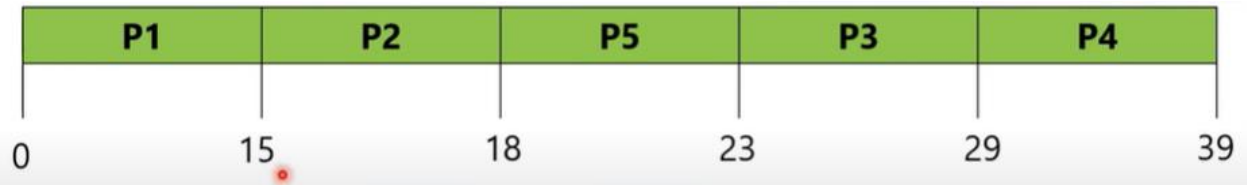
```
Please input number of Processes: 5
Input Process 1
PID: 1
Arrival Time: 0
Burst Time: 8
Input Process 2
PID: 2
Arrival Time: 2
Burst Time: 19
Input Process 3
PID: 3
Arrival Time: 4
Burst Time: 3
Input Process 4
PID: 4
Arrival Time: 5
Burst Time: 6
Input Process 5
PID: 5
Arrival Time: 7
Burst Time: 12

===== SJF Scheduling =====
Average Waiting Time: 9.40
Average Turnaround Time: 19.00
P1  P3  P4  P5  P2
0   8   11  17  29  48
```

**Test case 3**

| Process | Arrival Time | CPU Burst Time |
|---|---|---|
| P1 | 0 | 15 |
| P2 | 2 | 3 |
| P3 | 3 | 6 |

| | | |
|---|---|---|
| P4 | 6 | 10 |
| P5 | 7 | 5 |

### 1. SJF



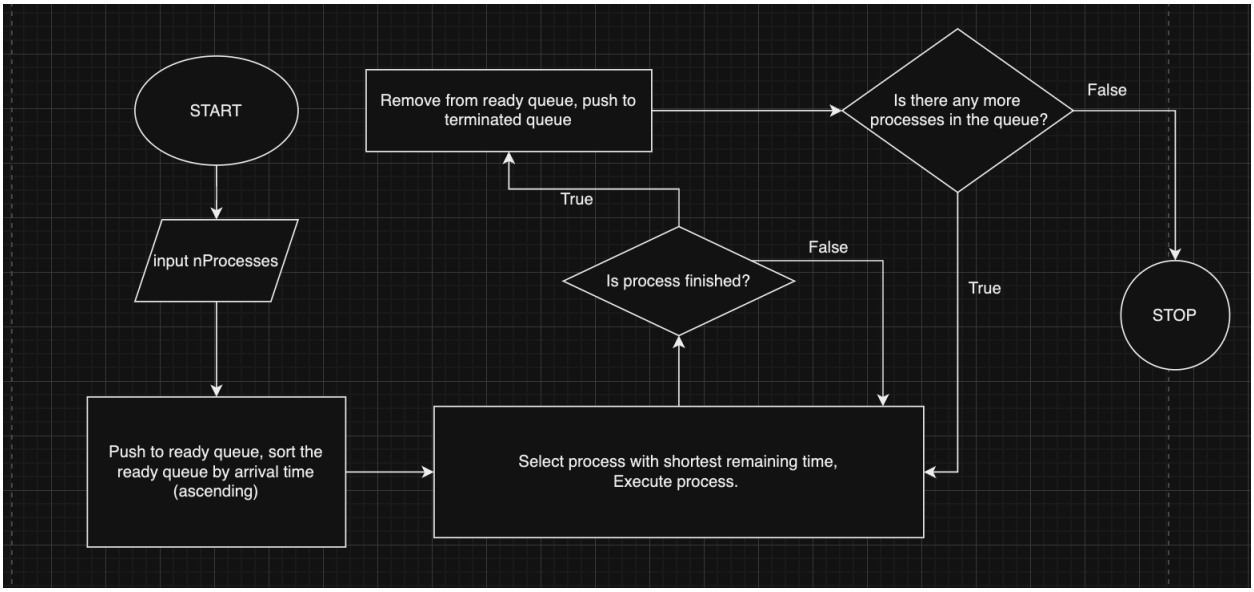

- Thời gian đợi trung bình: (0 + 13 + 20 + 23 + 11) / 5 = 13.4
- Thời gian hoàn thành trung bình: (15 + 16 + 26 + 33 + 16) / 5 = 21.2

```
Please input number of Processes: 5
Input Process 1
PID: 1
Arrival Time: 0
Burst Time: 15
Input Process 2
PID: 2
Arrival Time: 2
Burst Time: 3
Input Process 3
PID: 3
Arrival Time: 3
Burst Time: 6
Input Process 4
PID: 4
Arrival Time: 6
Burst Time: 10
Input Process 5
PID: 5
Arrival Time: 7
Burst Time: 5

===== SJF Scheduling =====
Average Waiting Time: 13.40
Average Turnaround Time: 21.20
P1  P2  P5  P3  P4
0   15  18  23  29  39
```

### 2. Giải thuật Shortest-Remaining-Time-First

Shortest-Remaining-Time-First

**Vẽ lưu đồ:**




**Chạy tay lưu đồ:**

1. Bắt đầu chương trình -> 2. Nhập vào n tiến trình -> 3. Đẩy vào Ready Queue (RQ), sắp xếp RQ theo thứ tự không giảm của Arrival Time -> 4. Chọn tiến trình có thời gian thực thi còn lại thấp nhất và thực thi trong 1 đơn vị thời gian -> 5. Kiểm tra tiến trình có hoàn thành chưa, nếu chưa về lại bước 4, nếu tiến trình đã hoàn thành -> 6. Xoá khỏi RQ, đẩy vào Terminated Queue -> Kiểm tra xem còn tiến trình nào trong RQ không, nếu còn thì quay lại bước 4, nếu không -> 7. Kết thúc chương trình.

**Code:**

```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
#define SORT_BY_REMAINING 4
typedef struct
{
    int iPID;
    int iArrival, iBurst;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
    int iRemaining, iLastRunning;
} PCB;

void inputProcess(int n, PCB P[])
```

```
{
    for(int i = 0 ; i < n;i++){
        printf("Input Process %d\n",i+1);
        printf("PID: ");
        scanf("%d",&P[i].iPID);
        printf("Arrival Time: ");
        scanf("%d",&P[i].iArrival);
        printf("Burst Time: ");
        scanf("%d",&P[i].iBurst);
        P[i].iRemaining = P[i].iBurst;
        P[i].iResponse = -1;
    }
}
void printProcess(int n, PCB P[])
{
    for(int i = 0 ; i < n;i++){
        printf("PID: %d\n",P[i].iPID);
        printf("Arrival Time: %d\n",P[i].iArrival);
        printf("Burst Time: %d\n",P[i].iBurst);
        printf("Start Time: %d\n",P[i].iStart);
        printf("Finish Time: %d\n",P[i].iFinish);
        printf("Waiting Time: %d\n",P[i].iWaiting);
        printf("Response Time: %d\n",P[i].iResponse);
        printf("Turn Around Time: %d\n",P[i].iTaT);

    }
}

void pushProcess(int *n, PCB P[], PCB Q)
{
    P[*n] = Q;
    (*n)++;
}

void removeProcess(int *n, int index, PCB P[]){
    for(int i = index ; i < *n - 1;i++){
        P[i] = P[i+1];
    }
    (*n)--;
}
void swapProcess(PCB *P, PCB *Q) {
    PCB temp = *P;
    *P = *Q;
    *Q = temp;
}
int partition (PCB P[], int low, int high, int iCriteria) {
    if (iCriteria == SORT_BY_ARRIVAL) {
```

```
    int pivot = P[high].iArrival;

    int i = (low-1);

    for(int j = low; j <= high; j++)
    {
        if (P[j].iArrival < pivot)
        {
            i++;
            swapProcess(&P[i], &P[j]);
        }
    }
    swapProcess(&P[i+1], &P[high]);
    return (i+1);
} else if (iCriteria == SORT_BY_PID) { // Sort by PID
    int pivot = P[high].iPID;

    int i = (low-1);

    for(int j = low; j <= high; j++)
    {
        if (P[j].iPID < pivot)
        {
            i++;
            swapProcess(&P[i], &P[j]);
        }
    }
    swapProcess(&P[i+1], &P[high]);
    return (i+1);
} else if (iCriteria == SORT_BY_BURST) {
    int pivot = P[high].iBurst;

    int i = (low-1);

    for(int j = low; j <= high; j++)
    {
        if (P[j].iBurst < pivot)
        {
            i++;
            swapProcess(&P[i], &P[j]);
        }
    }
    swapProcess(&P[i+1], &P[high]);
    return (i+1);
} else if (iCriteria == SORT_BY_START) {
    int pivot = P[high].iStart;
```

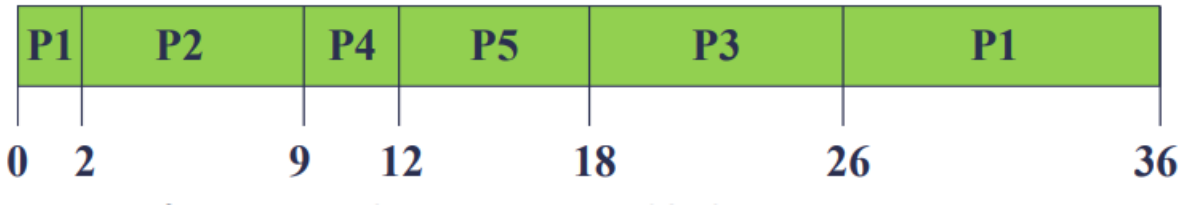
```
        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iStart < pivot)
            {
                i++;
                swapProcess(&P[i],&P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_REMAINING) {
        int pivot = P[high].iRemaining;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iRemaining < pivot)
            {
                i++;
                swapProcess(&P[i],&P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    }
}

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

    int pi = partition(P, low, high, iCriteria);

    quickSort(P, low, pi-1, iCriteria);
    quickSort(P, pi+1, high, iCriteria);
    }
}
void calculateAWT(int n, PCB P[])
{
    int sum = 0;
    for(int i = 0 ; i < n;i++){
        sum += P[i].iWaiting;
    }
    printf("Average Waiting Time: %f\n",(float)sum/n);
}
```

```
void calculateATaT(int n, PCB P[]){
    int sum = 0;
    for(int i = 0 ; i < n;i++){
        sum += P[i].iTaT;
    }
    printf("Average TAT Time: %f\n",(float)sum/n);
}
void ExportGanttChart(int a[], int totalTime) {
    printf("P%d\t", a[0]);
    for (int iTime = 1; iTime <= totalTime; ++iTime) {
        if (a[iTime] == a[iTime-1]) continue;
        printf("P%d\t", a[iTime]);
    }
    printf("\n0\t");
    for (int iTime = 1; iTime <= totalTime; ++iTime) {
        if (a[iTime] == a[iTime-1]) continue;
        printf("%d\t", iTime);
    }
    printf("%d\n", totalTime+1);
}


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);

    // SRT Algorithm Implementation


    // Push to ready queue
    int totalTime = 0;
    for (int i = 0; i < iNumberOfProcess; ++i) {
        totalTime += Input[0].iBurst;
        pushProcess(&iReady, ReadyQueue, Input[0]);
        removeProcess(&iRemain, 0, Input);
    }
```

```
    // Executing SRT Algorithm

    int ganttChart[totalTime];
    for (int currentTime = 0; currentTime <= totalTime-1; ++currentTime) {
        int i;
        for (i = 0; i < iReady; ++i) {
            if (ReadyQueue[i].iArrival <= currentTime) break;
        }
        // Pick the current process (which has a shortest remaining time)
        PCB* currentProcess = &ReadyQueue[i];
        // Store execution information to Gantt Chart
        ganttChart[currentTime] = currentProcess -> iPID;

        if (currentProcess -> iResponse == -1) {
            currentProcess -> iStart = currentTime;
            currentProcess -> iResponse = currentProcess -> iStart - currentProcess -> iArrival;
            currentProcess -> iWaiting = currentProcess -> iResponse;
            currentProcess -> iLastRunning = currentTime;
        }
        else {
            currentProcess -> iWaiting += currentTime - currentProcess -> iLastRunning - 1;
            currentProcess -> iLastRunning = currentTime;
        }
        currentProcess -> iRemaining--;
        if (currentProcess -> iRemaining == 0) {
            currentProcess -> iFinish = currentTime+1;
            currentProcess -> iTaT = currentTime+1 - currentProcess -> iArrival;
            pushProcess(&iTerminated, TerminatedArray, *currentProcess);
            removeProcess(&iReady, 0, ReadyQueue);
        }
        // Sort the ready queue
        quickSort(ReadyQueue, 0, iReady - 1, SORT_BY_REMAINING);
    }

    // Output

    printf("\n===== SRT Scheduling =====\n");
    ExportGanttChart(ganttChart, totalTime-1);
    quickSort(TerminatedArray, 0, iTerminated - 1,SORT_BY_PID);
    calculateAWT(iTerminated, TerminatedArray);
    calculateATaT(iTerminated, TerminatedArray);
    return 0;
}
```

**Test case 1: Chạy tay**

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



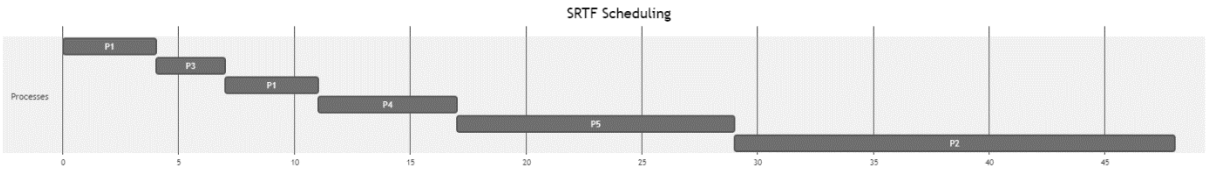

- Thời gian đợi trung bình: 7.4
- Thời gian hoàn thành trung bình: 14.6

**Chạy code:**

```
~/Desktop/UIT/OS/Thuc Hanh/Buoi 4
❯ ./SJF
Please input number of Process: 5
1 0 12
2 2 7
3 5 8
4 9 3
5 12 6
Input Process 1
PID: Arrival Time: Burst Time: Input Process 2
PID: Arrival Time: Burst Time: Input Process 3
PID: Arrival Time: Burst Time: Input Process 4
PID: Arrival Time: Burst Time: Input Process 5
PID: Arrival Time: Burst Time:
===== SRT Scheduling =====
P1      P2      P4      P5      P3      P1
0       2       9       12      18      26      36
Average Waiting Time: 7.400000
Average TAT Time: 14.600000
```

**Test case 2: Chạy tay**

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



- Thời gian đợi trung bình: (3 + 0 + 6 + 10 + 27) / 5 = 9.2
- Thời gian hoàn thành trung bình: (11 + 3 + 12 + 22 + 48) / 5 = 18.7999

**Chạy code:**

```
~/Desktop/UIT/OS/Thuc Hanh/Buoi 4
❯ ./SJF
Please input number of Process: 5
1 0 8
2 2 19
3 4 3
4 5 6
5 7 12
Input Process 1
PID: Arrival Time: Burst Time: Input Process 2
PID: Arrival Time: Burst Time: Input Process 3
PID: Arrival Time: Burst Time: Input Process 4
PID: Arrival Time: Burst Time: Input Process 5
PID: Arrival Time: Burst Time:
===== SRT Scheduling =====
P1      P3      P1      P4      P5      P2
0       4       7       11      17      29      48
Average Waiting Time: 9.200000
Average TAT Time: 18.799999
```

**Test case 3: Chạy tay**

| Process | Arrival Time | CPU Burst Time |
|---|---|---|
| P1 | 0 | 15 |
| P2 | 2 | 3 |
| P3 | 3 | 6 |
| P4 | 6 | 10 |
| P5 | 7 | 5 |


- Thời gian đợi trung bình: (24 + 0 + 2 + 10 + 4) / 5 = 8
- Thời gian hoàn thành trung bình: (39 + 3 + 8 + 20 + 9) / 5 = 15.8

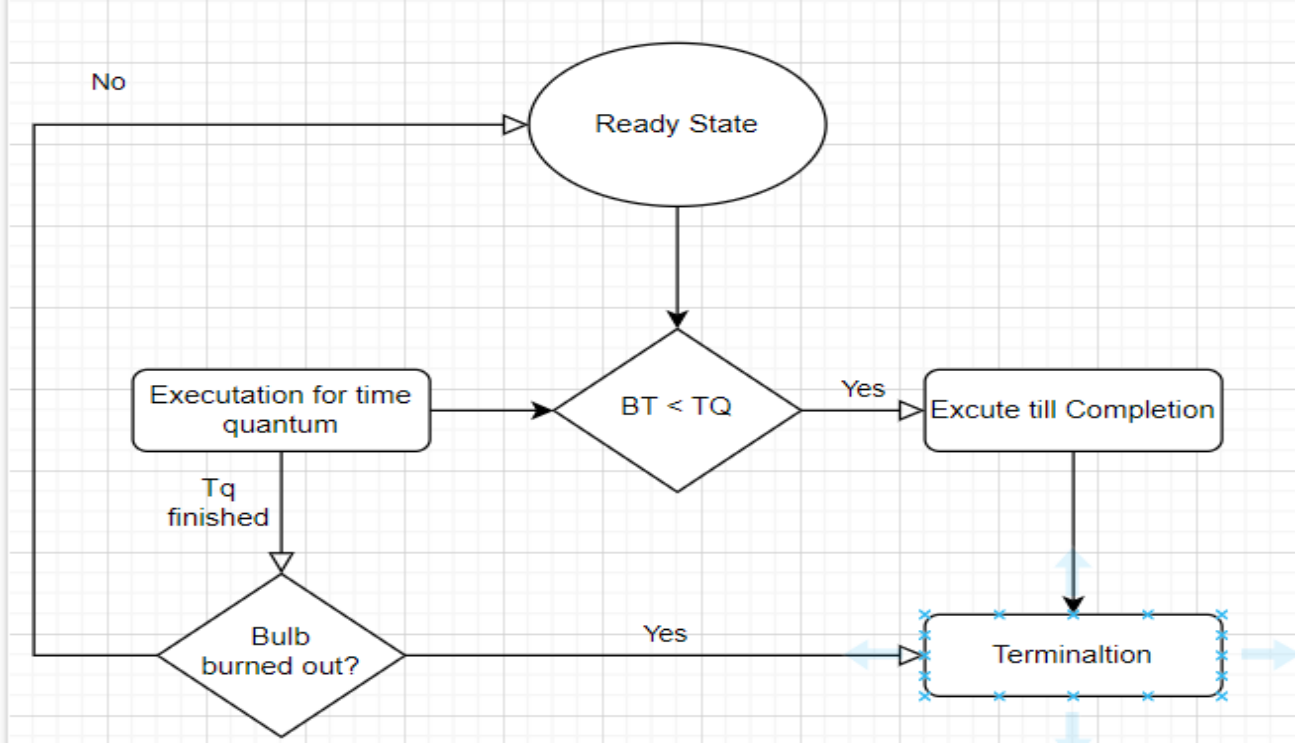
**Chạy code:**

```
❯ ./SJF
Please input number of Process: 5
1 0 15
2 2 3
3 3 6
4 6 10
5 7 5
Input Process 1
PID: Arrival Time: Burst Time: Input Process 2
PID: Arrival Time: Burst Time: Input Process 3
PID: Arrival Time: Burst Time: Input Process 4
PID: Arrival Time: Burst Time: Input Process 5
PID: Arrival Time: Burst Time:
===== SRT Scheduling =====
P1      P2      P3      P5      P4      P1
0       2       5       11      16      26      39
Average Waiting Time: 8.000000
Average TAT Time: 15.800000
❯
```

## 2.6. BÀI TẬP ÔN TẬP

### 1. Giải thuật Round Robin

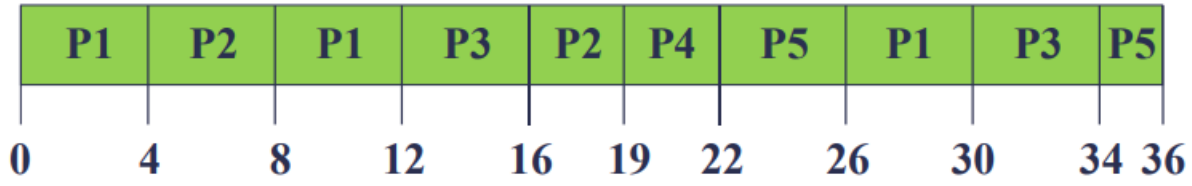
Trả lời...




**Test case 1:**

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

**RR (q = 4)**



- Thời gian đợi trung bình: 15.4
- Thời gian hoàn thành trung bình: 22.6

**Code:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#define SORT_BY_ARRIVAL 0
#define SORT_BY_PID 1
#define SORT_BY_TURN 2
#define SORT_BY_START 3
#define PRINT_DETAIL 1

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


void input(int numberOfProcess, PCB P[]) {
    for (int i = 0; i < numberOfProcess; i++) {
        printf("Enter Arrival Time for Process %d: ", i + 1);
        scanf("%d", &P[i].iArrival);

        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &P[i].iBurst);
        P[i].iBurst_have = P[i].iBurst;
        P[i].iPID = i + 1;
    }
}
```

```c
void print_Process(int numberofProcess, PCB P[], int option) {
    printf("\n");
    if(option == PRINT_DETAIL){
        printf("Process\t Start\t Finish Waiting Response   TaT\n");
        for (int i = 0; i<numberofProcess; i++)
        printf("P%d\t   %d\t   %d\t   %d\t     %d\t     %d\n", P[i].iPID , P[i].iStart , P[i].iFinish, P[i].iWaiting , P[i].iResponse , P[i].iTaT);
    }
    else {
        printf("Process\t   ArrivalTime\t   BurstTime\n");
        for (int i = 0; i < numberofProcess; i++)
        {
            printf("P%d\t\t%d\t\t%d\n", P[i].iPID , P[i].iArrival , P[i].iBurst);
        }
    }
    printf("\n");
}
```

```
void drawGanttChart(PCB Process, int current, bool option, int quantumTime){
        int runtime = 0;
        if(option==0) {
            while(runtime <= quantumTime){
                if(runtime == quantumTime/2)
                    printf("--P%d--", Process.iPID);
                else
                    printf("-");
                runtime++;
        }
        printf("%d", current);
        runtime=0;
        }
        if(option==1){
            while(runtime <= Process.iBurst_have)
            {
                if(runtime<= Process.iBurst_have/2)
                    printf("--P%d--", Process.iPID);
                else
                    printf("-");
                runtime++;
            }
            printf("%d", current);
            runtime = 0;
        }
}
```

```
void pushProcess(int *n, PCB P[], PCB Process) {
        P[(*n)++] = Process;
}

void removeProcess(int *numberOfProcess, int index, PCB P[]){
        //xóa phần tử và dồn các phần tử còn lại
        for (int i = index; i < *numberOfProcess -1; i++){
            P[i]=P[i+1];
        }
        (*numberOfProcess)--;
}

float calculateAWT(int numberOfProcess , PCB P[]) {
    int totalWT = 0;
    for(int i = 0; i < numberOfProcess; i++)
        totalWT += P[i].iWaiting;

    return (float)totalWT/numberOfProcess;
}

float calculateATaT(int numberOfProcess , PCB P[]) {
    int totalTaT = 0;
    for(int i = 0; i < numberOfProcess; i++)
        totalTaT += P[i].iTaT;

    return (float)totalTaT/numberOfProcess;
}
```

```
void interchangeSort (PCB P[ ], int start, int end, int criteria ) {

    if(criteria == SORT_BY_ARRIVAL) {
        for(int i = start; i < end ; i++)
        {
            for(int j = i + 1; j <= end ; j++)
            {
                if(P[i].iArrival > P[j].iArrival)
                {
                    PCB tempProcess = P[j];
                    P[j] = P[i];
                    P[i] = tempProcess;
                }
            }
        }
    }
    if(criteria == SORT_BY_PID) {
        for(int i = start; i < end ; i++)
        {
            for(int j = i + 1; j <= end ; j++)
            {
                if(P[i].iPID > P[j].iPID)
                {
                    PCB tempProcess = P[j];
                    P[j] = P[i];
                    P[i] = tempProcess;
                }
            }
        }
    }
```

```
int current = 0;
bool sort_rq = 1;

int main() {
    int iNumberOfProcess;
    printf("Enter the number of processes: ");
    scanf("%d", &iNumberOfProcess);

    int quantumTime;
    printf("Enter the quantum time: ");
    scanf("%d", &quantumTime);

    PCB Input[iNumberOfProcess];
    PCB ReadyQueue[iNumberOfProcess];
    PCB TerminatedArray[iNumberOfProcess];
    int iRemain = iNumberOfProcess;
    int iReady = 0, iTerminated = 0;

    input(iNumberOfProcess, Input);
```

```
    interchangeSort(Input, 0, iNumberOfProcess - 1, SORT_BY_ARRIVAL);
    print_Process(iRemain , Input, 0);

    pushProcess(&iReady , ReadyQueue , Input[0]);
    removeProcess(&iRemain , 0 ,Input);
    int runtime = 0;
    printf(".....GanttChart RR.....\n");

    if(ReadyQueue[0].iBurst_have > quantumTime)
    {
        ReadyQueue[0].iBurst_have -= quantumTime;
        ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
        ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;
        current = ReadyQueue[0].iStart + quantumTime;

        if(ReadyQueue[0].iStart != current)
            printf("  %d" , ReadyQueue[0].iStart);

        drawGanttChart(ReadyQueue[0], current, 0, quantumTime);
    }
    else
    {
        ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
        ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst_have;
        ReadyQueue[0].iResponse = ReadyQueue[0].iArrival;
        current= ReadyQueue[0].iFinish;
        ReadyQueue[0].iBurst_have = 0;
        sort_rq = 0;

        if(ReadyQueue[0].iStart != current)
            printf("  %d" , ReadyQueue[0].iStart);
```

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

        if(iReady > 0)
        {
            if(ReadyQueue[0].iBurst_have > 0)
            {
                if(sort_rq == 1 && iReady > 1)
                {
                    interchangeSort(ReadyQueue, 0, iReady - 1, SORT_BY_TURN);
                }

                if(ReadyQueue[0].iBurst_have == ReadyQueue[0].iBurst)
                {
                    ReadyQueue[0].iStart = current;
                    ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;
                }

                if(ReadyQueue[0].iBurst_have > quantumTime)
                {
                    ReadyQueue[0].iBurst_have -= quantumTime;
                    current += quantumTime;
                    sort_rq = 1;
```

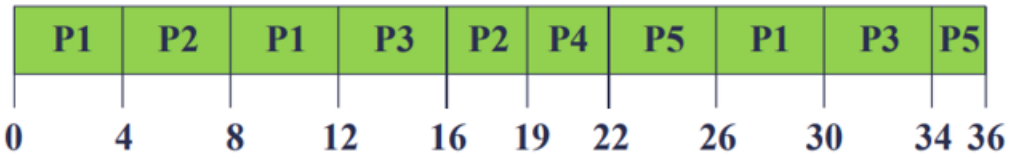
```
                    drawGanttChart(ReadyQueue[0], current, 0, quantumTime);
                }
                else
                {
                    ReadyQueue[0].iFinish = current + ReadyQueue[0].iBurst_have;
                    current = ReadyQueue[0].iFinish;
                    ReadyQueue[0].iBurst_have = 0;
                    sort_rq = 0;

                    drawGanttChart(ReadyQueue[0], current, 1, quantumTime);

                }

            }
            else
            {
                ReadyQueue[0].iTaT = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival;
                ReadyQueue[0].iWaiting = ReadyQueue[0].iTaT - ReadyQueue[0].iBurst;
                pushProcess(&iTerminated, TerminatedArray, ReadyQueue[0]);
                removeProcess(&iReady, 0, ReadyQueue);
                continue;
            }
        }

        else
        {
            current = Input[0].iArrival;
            printf("  %d" , current);
            pushProcess(&iReady , ReadyQueue , Input[0]);
            removeProcess(&iRemain , 0 ,Input);
        }
```

```
58
59          else
60          {
61              current = Input[0].iArrival;
62              printf("  %d" , current);
63              pushProcess(&iReady , ReadyQueue , Input[0]);
64              removeProcess(&iRemain , 0 ,Input);
65          }
66
67
68      }
69
70      interchangeSort(TerminatedArray, 0 , iTerminated - 1, SORT_BY_PID);
71      print_Process(iTerminated, TerminatedArray, PRINT_DETAIL);
72      printf("AWT: %.2f\n", calculateAWT(iTerminated, TerminatedArray));
73      printf("ATaT: %.2f", calculateATaT(iTerminated, TerminatedArray));
74
75      return 0;
76  }
77
```

**Test case 1:**

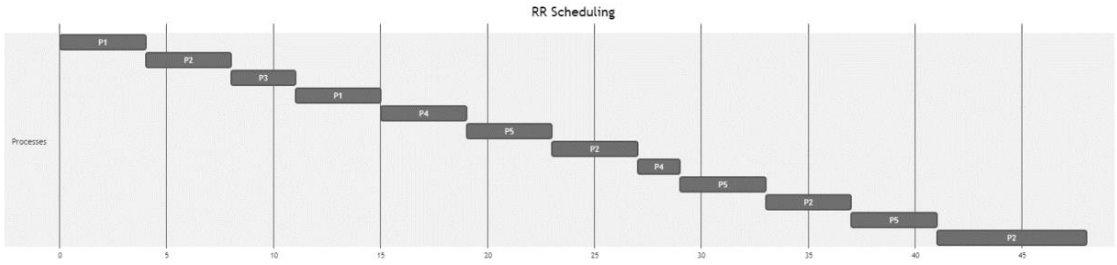
```
Process    ArrivalTime     BurstTime
P1              0               12
P2              2               7
P3              5               8
P4              9               3
P5              12              6

.....GanttChart RR.....
  0---P1---4---P2---8---P1---12---P3---16-P2-19-P4-22---P5---26-P1-30-P3-34-P5-36
Process  Start   Finish Waiting Response   TaT
P1         0       30      18        0       30
P2         4       19      10        2       17
P3         12      34      21        7       29
P4         19      22      10        10      13
P5         22      36      18        10      24

AWT: 15.40
ATaT: 22.60
```

3. **RR (q = 4)**




- Thời gian đợi trung bình: 15.4
- Thời gian hoàn thành trung bình: 22.6

**Test case 2:**

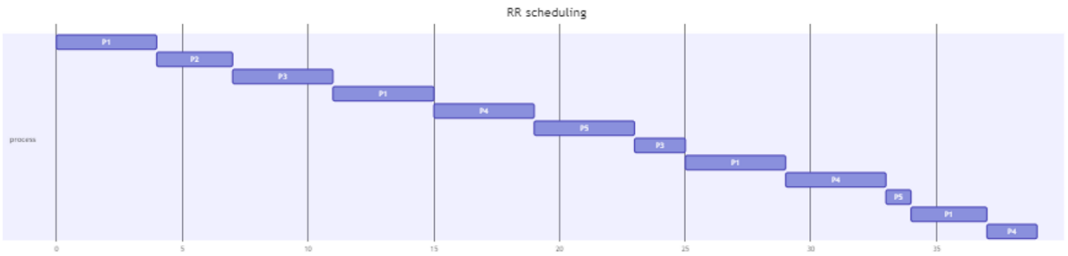



- Thời gian đợi trung bình: (7 + 27 + 4 + 18 + 22) / 5 = 15.6
- Thời gian hoàn thành trung bình: (15 + 48 + 11 + 29 + 41) / 5 = 25.2

```
Process    ArrivalTime     BurstTime
P1               0               8
P2               2               19
P3               4               3
P4               5               6
P5               7               12

.....GanttChart RR.....
  0---P1---4---P2---8-P3-11-P1-15---P4---19---P5--
---P2---37-P5-41---P2---45-P2-48
Process  Start   Finish Waiting Response   TaT
P1         0       15      7         0       15
P2         4       48      27        2       46
P3         8       11      4         4       7
P4         15      29      18        10      24
P5         19      41      22        12      34

AWT: 15.60
ATaT: 25.20
```

Test case 3:

```
Process    ArrivalTime     BurstTime
P1              0               15
P2              2               3
P3              3               6
P4              6               10
P5              7               5

.....GanttChart RR.....
  0---P1---4-P2-7---P3---11---P1---15---P4---19---P5---23-P3-25---P1---29---P4-
--33-P5-34-P1-37-P4-39
Process  Start   Finish Waiting Response   TaT
P1         0       37      22        0       37
P2         4       7       2         2       5
P3         7       25      16        4       22
P4         15      39      23        9       33
P5         19      34      22        12      27

AWT: 17.00
ATaT: 24.80
```




- Thời gian đợi trung bình: (22 + 2 + 16 + 23 + 22) / 5 = 17
- Thời gian hoàn thành trung bình: (37 + 5 + 22 + 33 + 27) / 5 = 24.8