|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NHÓM 2**   |  |  | | --- | --- | | Nguyễn Trần Bảo Anh | 22520066 | | Trần Đồng Trúc Lam | 22520746 | | Trần Mạnh Kiên | 22520711 | | Hồ Văn Trí | 23521637 |   Lớp: IT007.O21.CNVN |

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:**

**2.5. BÀI TẬP THỰC HÀNH**

# Giải thuật Shortest-Job-First

Trả lời:

# 1.1. Flowchart giải thuật

A black and white diagram

Description automatically generated

# 1.2. Code của giải thuật

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <iomanip>  #include <string.h>  using namespace std;  struct process  {  char pid;  int arrival\_time;  int burst\_time;  int start\_time;  int finish\_time;  int turnaround\_time;  int waiting\_time;  int response\_time;  };  bool compareArrival(process p1, process p2)  {  return p1.arrival\_time < p2.arrival\_time;  }  int main()  {  int n;  struct process p[100];  float avg\_turnaround\_time;  float avg\_waiting\_time;  float avg\_response\_time;  int total\_turnaround\_time = 0;  int total\_waiting\_time = 0;  int total\_response\_time = 0;  int current\_time = 0;  int completed = 0;  int is\_completed[100];  memset(is\_completed, 0, sizeof(is\_completed));  cout << setprecision(2) << fixed;  cout << "\n\nEnter the number of processes: ";  cin >> n;  for (int i = 0; i < n; i++)  {  // Assigning PID  char pid;  if (i < 26)  pid = 'A' + i;  else  pid = 'A' + (i - 26);  p[i].pid = pid;  cout << "\nEnter arrival time of process " << p[i].pid << ": ";  cin >> p[i].arrival\_time;  cout << "Enter burst time of process " << p[i].pid << ": ";  cin >> p[i].burst\_time;  }  while (completed != n)  {  int idx = -1;  int mn = 10000000;  for (int i = 0; i < n; i++)  {  // Check if it's arrrived and has not been completed yet  if (p[i].arrival\_time <= current\_time && is\_completed[i] == 0)  {  if (p[i].burst\_time < mn)  {  mn = p[i].burst\_time;  idx = i;  }  // Handling multiple processes have the same burst time, choose based on its arrival time  if (p[i].burst\_time == mn)  {  if (p[i].arrival\_time < p[idx].arrival\_time)  {  mn = p[i].burst\_time;  idx = i;  }  }  }  }  if (idx != -1)  {  p[idx].start\_time = current\_time;  p[idx].finish\_time = p[idx].start\_time + p[idx].burst\_time;  p[idx].turnaround\_time = p[idx].finish\_time - p[idx].arrival\_time;  p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;  p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;  total\_turnaround\_time += p[idx].turnaround\_time;  total\_waiting\_time += p[idx].waiting\_time;  total\_response\_time += p[idx].response\_time;  is\_completed[idx] = 1;  completed++;  current\_time = p[idx].finish\_time;  }  else  {  current\_time++;  }  }  avg\_turnaround\_time = (float)total\_turnaround\_time / n;  avg\_waiting\_time = (float)total\_waiting\_time / n;  avg\_response\_time = (float)total\_response\_time / n;  sort(p, p + n, compareArrival);  printf("\n\n ===== SJF Scheduling ===== \n");  cout << "#P\t"  << "AT\t"  << "BT\t"  << "ST\t"  << "FT\t"  << "TAT\t"  << "WT\t"  << "RT\t"  << "\n";  for (int i = 0; i < n; i++)  {  cout << p[i].pid << "\t" << p[i].arrival\_time << "\t" << p[i].burst\_time << "\t" << p[i].start\_time << "\t" << p[i].finish\_time << "\t" << p[i].turnaround\_time << "\t" << p[i].waiting\_time << "\t" << p[i].response\_time << "\t"  << "\n";  }  cout << "\nAverage Turnaround Time = " << avg\_turnaround\_time << endl;  cout << "Average Waiting Time = " << avg\_waiting\_time << endl;  cout << "Average Response Time = " << avg\_response\_time << endl;  }  /\*  AT - Arrival Time của tiến trình  BT - Burst time của tiến trình  ST - Thời gian bắt đầu của tiến trình  FT - Thời gian kết thúc của tiến trình  TAT - Thời gian hoàn thành của tiến trình  WT - Thời gian đợi của tiến trình  RT - Thời gian đáp ứng của tiến trình  Công thức sử dụng:  TAT = FT - AT  WT = TAT - BT (= FT - AT - BT)  RT = ST - AT  \*/ |

# 1.3. Testcases

# Ví dụ 1:

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

* Kết quả khi chạy code

A screenshot of a computer program

Description automatically generated

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

* + - Kết quả khi giải tay

Graphical user interface, application

Description automatically generated

Hình 2:Kết quả khi giải tay ví dụ 1 bằng giải thuật SJF

1. Ví dụ 2:

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

* + 1. Kết quả khi chạy code

A screenshot of a computer program

Description automatically generated

Hình 3: Kết quả khi giải ví dụ 2 bằng code giải thuật SJF

* + 1. Kết quả khi giải tay

A picture containing text

Description automatically generated

Hình 4:Kết quả khi giải tay ví dụ 2 bằng giải thuật SJF

* 1. Ví dụ 3:

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

* + 1. Kết quả khi chạy code

A screenshot of a computer program

Description automatically generated

Hình 5: Kết quả khi giải ví dụ 3 bằng code giải thuật SJF

* + 1. Kết quả khi giải tay

Diagram

Description automatically generated

Hình 6: Kết quả khi giải tay ví dụ 3 bằng giải thuật SJF

# Giải thuật Shortest-Remaining-Time-First

**2.1/ Lưu đồ giải thuật**

A black background with white squares

Description automatically generated

**2.2/ Chạy tay lưu đồ giải thuật**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Waiting | Response | Turnaround |
| P1 | 2 | 6 | 7 | 1 | 13 |
| P2 | 5 | 2 | 0 | 0 | 2 |
| P3 | 1 | 8 | 14 | 14 | 22 |
| P4 | 0 | 3 | 0 | 0 | 3 |
| P5 | 4 | 4 | 2 | 0 | 6 |

Average Turn Around Time = 9.2

Average Waiting Time = 4.6

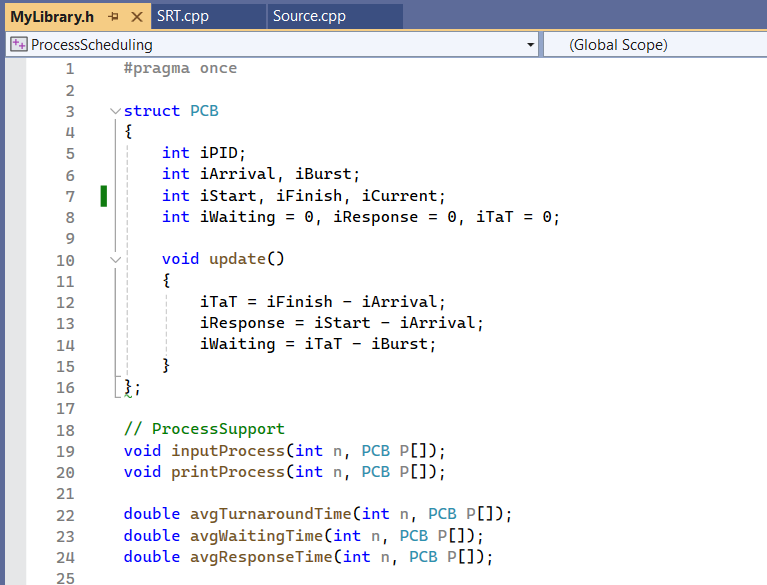
Average Response Time = 3

A green rectangular object with black text

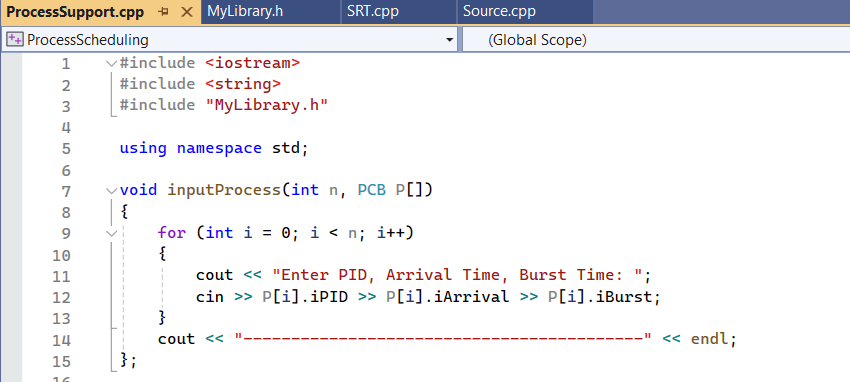
Description automatically generated

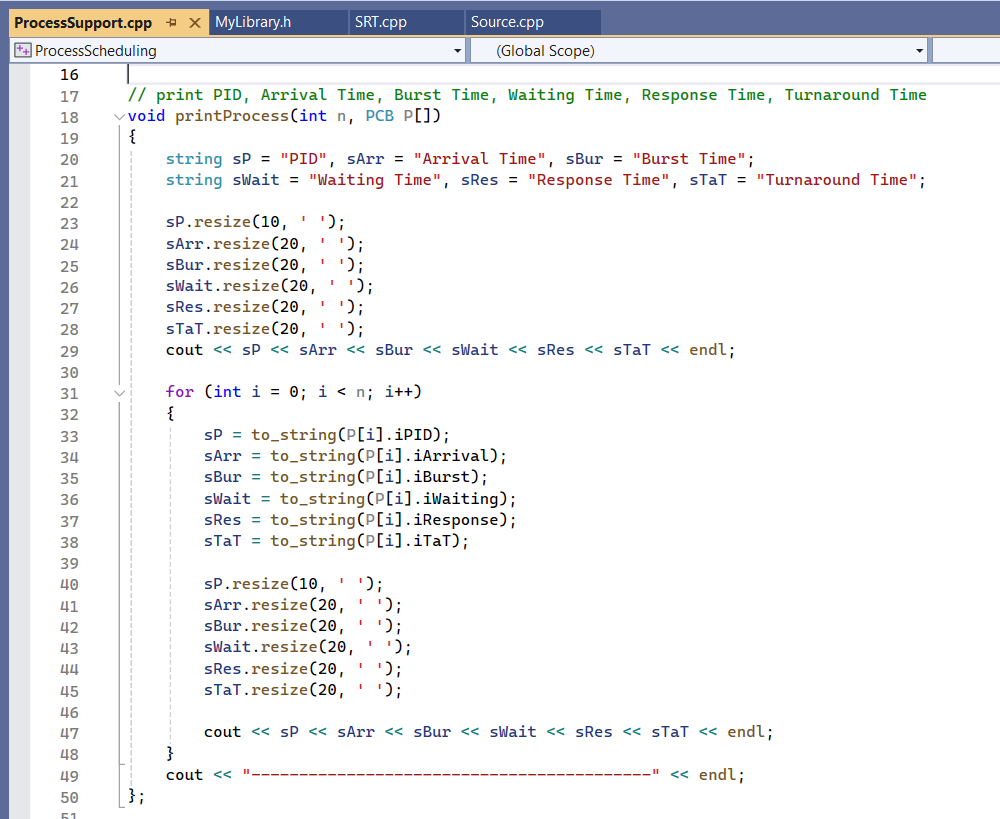
**2.3/ Hiện thực code**

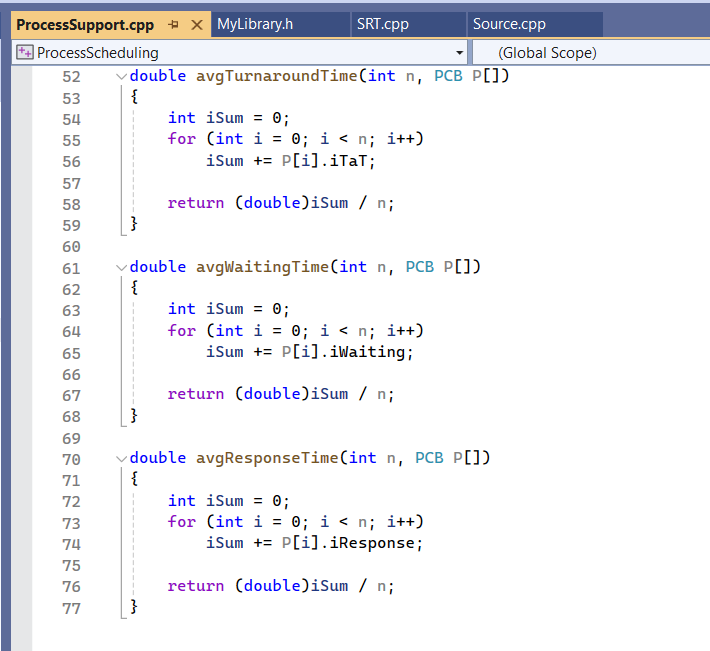
File header



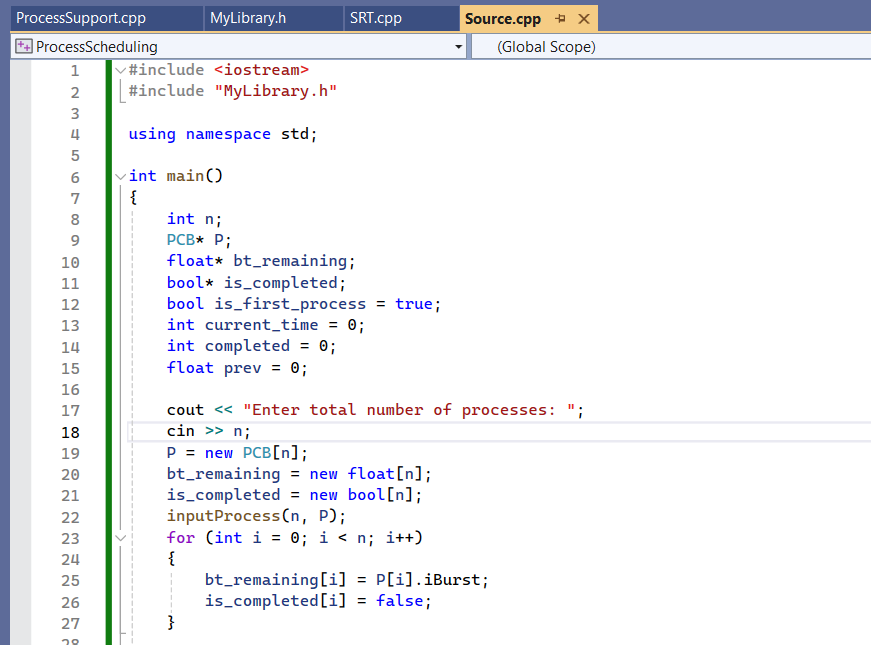
File Cpp

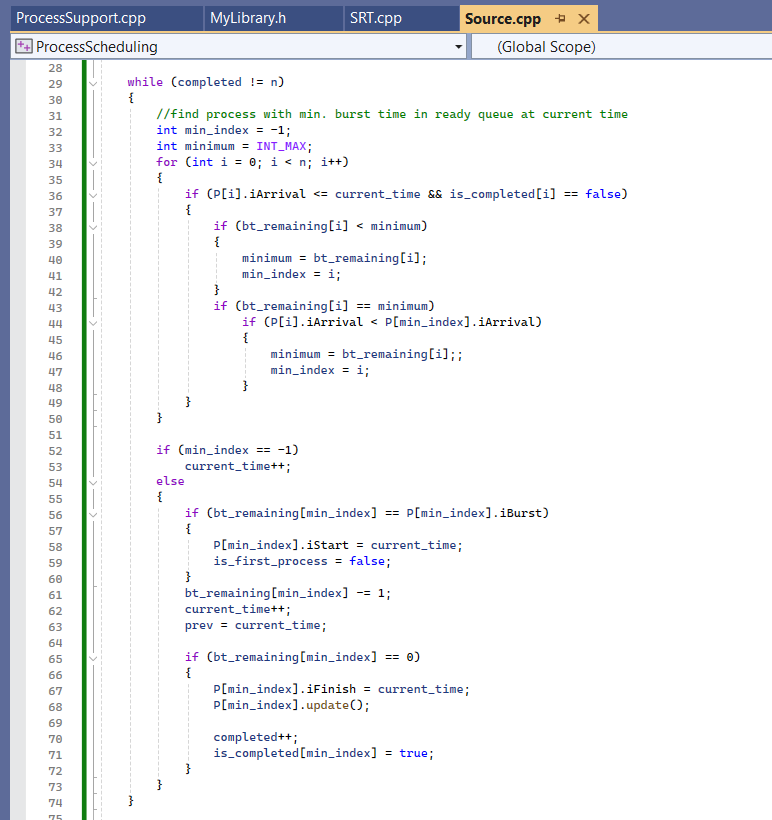


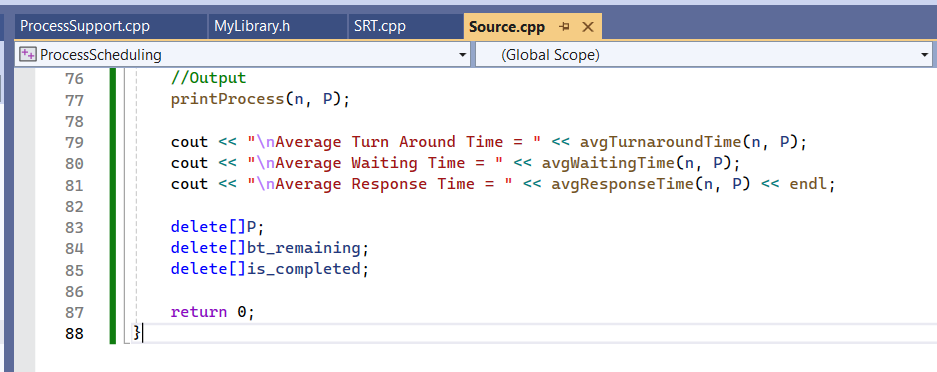




File main







**2.4/ Chạy code và kiểm chứng**

**Test 1**

A screenshot of a computer

Description automatically generated

Giải tay :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P4 | P3 | P0 |

0 5 14 17 24 36

**Test 2**

A screenshot of a computer

Description automatically generated

Giải tay :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P4 | P2 | P5 | P3 |

0 5 10 14 20 29

**Test 3**

A screenshot of a computer

Description automatically generated

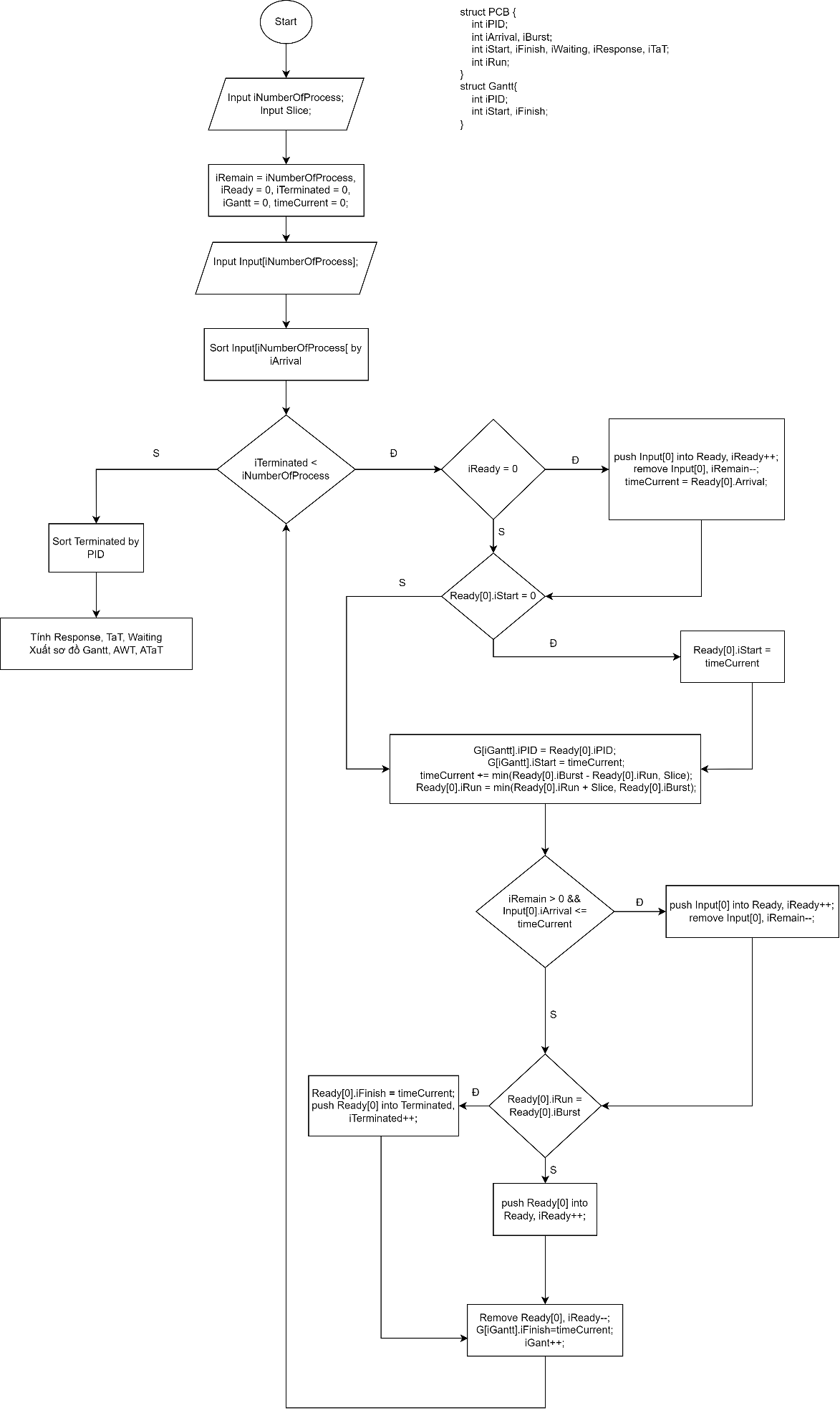
Giải tay :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P3 | P1 | P2 | P4 | P5 |

0 7 10 18 22 31

# Giải thuật Round Robin

3.1/ Lưu đồ giải thuật



3.2/ Chạy tay lưu đồ

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 2 | 11 |
| P2 | 0 | 8 |
| P3 | 7 | 5 |
| P4 | 9 | 10 |
| P5 | 13 | 6 |

Với quantum time = 6

Kết quả chạy tay:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P2 | P1 | P2 | P3 | P4 | P1 | P5 | P4 |
|  |  |  |  |  |  |  |  |

0 6 12 14 19 25 30 36 40

Thời gian chờ trung bình: (17+6+7+21+17)/5 = 13.6

Thời gian hoàn thành trung bình: (28+14+12+31+23) = 108/5 = 21.6

3.3/ Code giải thuật

#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 PCB {

    int iPID;

    int iArrival, iBurst;

    int iStart, iFinish, iWaiting, iResponse, iTaT;

    int iRun;

} PCB;

typedef struct Gantt{

    int iPID;

    int iStart, iFinish;

} Gantt;

void inputProcess(int n, PCB P[])

{

    for(int i=0; i<n; i++)

    {

        scanf("%d %d %d", &P[i].iPID, &P[i].iArrival, &P[i].iBurst);

        P[i].iRun = 0;

    }

}

void printProcess(int n, PCB P[])

{

    for(int i=0; i<n; i++)

        printf("%d %d %d\n",P[i].iPID, P[i].iArrival, P[i].iBurst);

}

void exportGanttChart(int n, Gantt P[])

{

    for(int i=0; i<n; i++)

    {

        printf("%d->%d: P%d\n", P[i].iStart, P[i].iFinish, P[i].iPID);

    }

}

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

{

    P[\*n]=Q;

    (\*n)++;

}

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

{

    for(int i=index+1; i < (\*n); i++)

        P[i-1]=P[i];

    (\*n)--;

}

void swapProcess(PCB \*P, PCB \*Q)

{

    PCB temp=\*P;

    \*P=\*Q;

    \*Q=temp;

}

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

{

    int pivot=high, i=low - 1;

    for(int j=low; j<=high; j++)

    {

        switch (iCriteria)

        {

        case 0:

            if(P[j].iArrival < P[pivot].iArrival)

            {

                i++;

                swapProcess(&P[i], &P[j]);

            }

        break;

        case 1:

            if(P[j].iPID < P[pivot].iPID)

            {

                i++;

                swapProcess(&P[i], &P[j]);

            }

        break;

        }

    }

    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;

    float avg;

    for(int i=0; i<n; i++)

    {

        sum+=P[i].iWaiting;

    }

    avg=sum/(n\*1.0);

    printf("Average waiting time: %f \n", avg);

}

void calculateATaT(int n, PCB P[])

{

    int sum=0;

    float avg;

    for(int i=0; i<n; i++)

    {

        sum+=P[i].iTaT;

    }

    avg=sum/(n\*1.0);

    printf("Average turn around time: %f \n", avg);

}

void calculateTime(int n, PCB P[])

{

    for(int i = 0; i < n; i++)

    {

        P[i].iResponse = P[i].iStart - P[i].iArrival;

        P[i].iTaT = P[i].iFinish - P[i].iArrival;

        P[i].iWaiting = P[i].iTaT - P[i].iBurst;

    }

}

int min(int x, int y)

{

    return(x < y? x : y);

}

int main()

{

    PCB Input[10];

    PCB Ready[10];

    PCB Terminated[10];

    Gantt G[100];

    int iNumberOfProcess, Slice;

    printf("Input number of Process: ");

    scanf("%d", &iNumberOfProcess);

    printf("Input slice time: ");

    scanf("%d", &Slice);

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

    inputProcess(iNumberOfProcess, Input);

    quickSort(Input, 0, iNumberOfProcess-1, SORT\_BY\_ARRIVAL);

    while (iTerminated < iNumberOfProcess)

    {

        if(iReady == 0)

        {

            pushProcess(&iReady, Ready, Input[0]);

            removeProcess(&iRemain, 0, Input);

            timeCurrent = Ready[0].iArrival;

        }

        if(Ready[0].iStart == 0) Ready[0].iStart = timeCurrent;

        G[iGantt].iPID = Ready[0].iPID;

        G[iGantt].iStart = timeCurrent;

        timeCurrent += min(Ready[0].iBurst - Ready[0].iRun, Slice);

        Ready[0].iRun = min(Ready[0].iRun + Slice, Ready[0].iBurst);

        while(iRemain > 0 && Input[0].iArrival <= timeCurrent)

        {

            pushProcess(&iReady, Ready, Input[0]);

            removeProcess(&iRemain, 0, Input);

        }

        if(Ready[0].iRun == Ready[0].iBurst)

        {

            Ready[0].iFinish = timeCurrent;

            pushProcess(&iTerminated, Terminated, Ready[0]);

        }

        else pushProcess(&iReady, Ready, Ready[0]);

        removeProcess(&iReady, 0, Ready);

        G[iGantt].iFinish = timeCurrent;

        iGantt++;

    }

    quickSort(Terminated, 0, iTerminated - 1, SORT\_BY\_PID);

    calculateTime(iTerminated, Terminated);

    printf("\n===== RR Scheduling =====\n");

    exportGanttChart(iGantt, G);

    calculateAWT(iTerminated, Terminated);

    calculateATaT(iTerminated, Terminated);

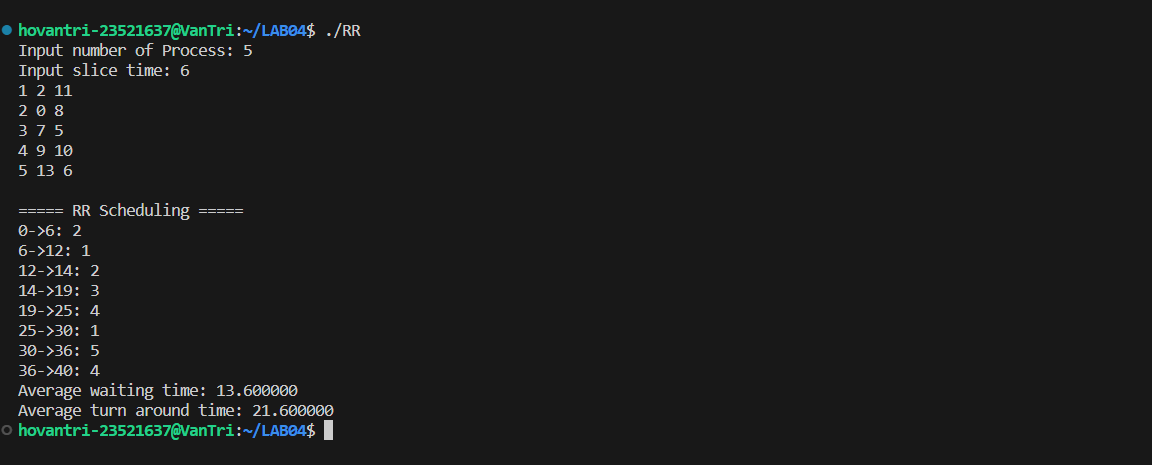
    return 0;

}

3.4/ Kiểm chứng code

VD 1: ví dụ đã chạy bằng lưu đồ ở trên

Kết quả khi chạy code:



VD 2:

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

Với quantum time = 4

Kết quả chạy tay:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P1 | P3 | P2 | P4 | P5 | P1 | P3 | P5 |
|  |  |  |  |  |  |  |  |  |  |

0 4 8 12 16 19 22 26 30 34 36

Thời gian chờ trung bình: (18+10+21+10+18)/5 = 15.4

Thời gian hoàn thành trung bình: (40+17+29+13+24)/5 = 22.6

Kết quả chạy code:



VD 3:

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 3 | 5 |
| P2 | 4 | 6 |
| P3 | 1 | 3 |
| P4 | 0 | 1 |
| P5 | 18 | 5 |

Với quantum time = 4

Kết quả chạy tay:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P4 | P3 | P1 | P2 | P1 | P2 | Trống | P5 |
|  |  |  |  |  |  |  |  |

0 1 4 8 12 13 15 18 23

Thời gian chờ trung bình: (5+5+0+0+0)/5 = 2

Thời gian hoàn thành trung bình: (10+11+3+1+5)/5 = 6

Kết quả chạy code:

