Lab

4

BÁO CÁO BÀI THỰC HÀNH SỐ 4

LẬP LỊCH TIẾN TRÌNH

Môn học: Hệ điều hành (IT007)

Sinh viên thực hiện	Nguyễn Đức Tấn
Thời gian thực hiện	27/04/2024

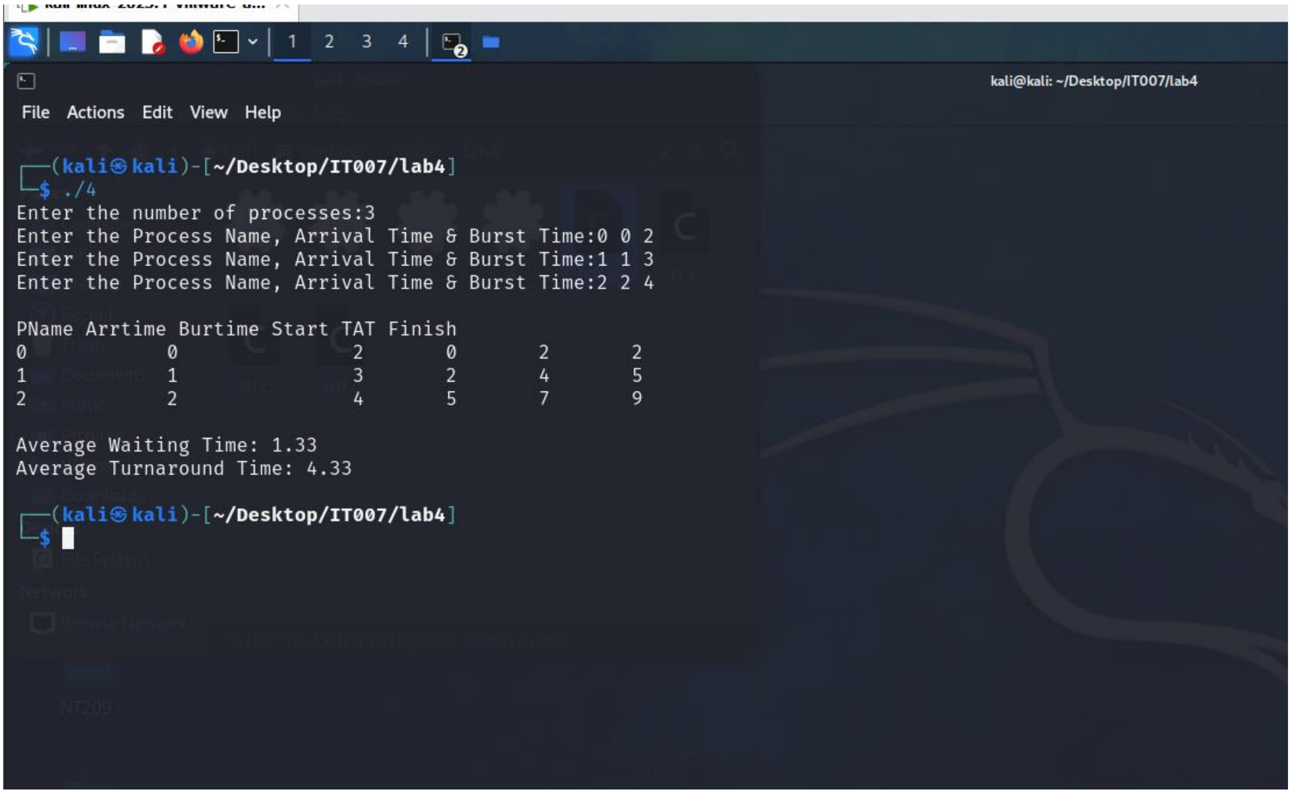
THỰC HÀNH:

0- GIẢI THUẬT FCFS

a. Source code:

```
#include <stdio.h>
void main()
{
    int pn[10];
    int arr[10], bur[10], star[10], finish[10], tat[10], wt[10], i, n;
    int totwt = 0, tottat = 0;
    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", &pn[i], &arr[i], &bur[i]);
    }
    for (i = 0; i < n; i++)
    {
        if (i == 0)
        {
            star[i] = arr[i];
            wt[i] = star[i] - arr[i];
            finish[i] = star[i] + bur[i];
            tat[i] = finish[i] - arr[i];
        }
        else
        {
            star[i] = finish[i - 1];
            wt[i] = star[i] - arr[i];
            finish[i] = star[i] + bur[i];
            tat[i] = finish[i] - arr[i];
        }
    }
    printf("\nPName Arrtime Burtime Start TAT Finish");
    for (i = 0; i < n; i++)
    {
        printf("\n%d\t%6d\t\t%6d\t%6d\t%6d\t%6d", pn[i], arr[i], bur[i], star[i],
tat[i], finish[i]);
        totwt += wt[i];
        tottat += tat[i];
    }

    float avg_wt = (float) totwt / n;
    float avg_tat = (float) tottat / n;

    printf("\n\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f", avg_tat);
}
```

b. Thực thi chương trình:



```
┌──(kali㉿kali)-[~/Desktop/IT007/lab4]
└─$ ./4
Enter the number of processes:3
Enter the Process Name, Arrival Time & Burst Time:0 0 2
Enter the Process Name, Arrival Time & Burst Time:1 1 3
Enter the Process Name, Arrival Time & Burst Time:2 2 4

PName Arrtime Burtime Start TAT Finish
0       0       2       0       2       2
1       1       3       2       4       5
2       2       4       5       7       9

Average Waiting Time: 1.33
Average Turnaround Time: 4.33

┌──(kali㉿kali)-[~/Desktop/IT007/lab4]
└─$ 
```

BÀI TẬP ÔN TẬP

A – GIẢI THUẬT SJF

1. Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
typedef struct {
    int iArrival, iBurst, iPID;
    int iResponse, iWaiting, iStart, iFinish, iTaT;
}PCB;

      void INIT_ARR(int Num_Of_Process, PCB P[]){
          for(int i=0; i<Num_Of_Process; i++){
              P[i].iPID = i + 1;
              printf("Process %d: \n", i+1);
              printf("Arrival: ");
              scanf("%d", &P[i].iArrival);
              printf("Burst: ");
              scanf("%d", &P[i].iBurst);
          }
      }

      void PRINT_ARR(int Num_Of_Process, PCB P[]){
          for(int i=0; i<Num_Of_Process; i++){
              printf("P%d ", P[i].iPID);
```

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

void swap(PCB *A, PCB *B){
    PCB Temp = *A;
    *A = *B;
    *B = Temp;
}

void quickSort_Arrival(PCB a[], int l, int r){
    PCB p=a[(l+r)/2]; ///p: la phan tu privot
    int i=l, j=r;
    while(i<=j){
        while(a[i].iArrival < p.iArrival){
            i++;
        }
        while(a[j].iArrival > p.iArrival){
            j--;
        }
        if(i<=j){
            swap(&a[i],&a[j]);
            i++;
            j--;
        }
    }
    if(i<r){
        quickSort_Arrival(a,i,r);
    }
    if(l<j){
        quickSort_Arrival(a,l,j);
    }
}

void quickSort_Burst(PCB a[], int l, int r){
    PCB p=a[(l+r)/2]; ///p: la phan tu privot
    int i=l, j=r;
    while(i<=j){
        while(a[i].iBurst<p.iBurst){
            i++;
        }
        while(a[j].iBurst>p.iBurst){
            j--;
        }
        if(i<=j){
            swap(&a[i],&a[j]);
            i++;
            j--;
        }
    }
    if(i<r){
        quickSort_Burst(a,i,r);
    }
    if(l<j){
        quickSort_Burst(a,l,j);
    }
}

void UPDATE_START(int P_Terminated, PCB Terminated_Arr[], PCB *Q, int *Time_line){
    if(P_Terminated == 0){
        Q->iStart = Q->iArrival;
    }
    else if(P_Terminated > 0){
        if(Q->iArrival <= Terminated_Arr[P_Terminated-1].iFinish){
            Q->iStart = Terminated_Arr[P_Terminated-1].iFinish;
        }
        else if(Q->iArrival > Terminated_Arr[P_Terminated-1].iFinish){
            Q->iStart = Q->iArrival;
        }
    }
    *Time_line = Q->iStart;
}

void PUSH_PROCESS(int *n, PCB P[], PCB Q){
    P[*n] = Q;
    *n = *n + 1;
}

void REMOVE_PROCESS(int *n, int index, PCB P[]){
    for(int i=index; i<*n-1; i++){
        swap(&P[i], &P[i+1]);
    }
    *n = *n - 1;
}

void PRINT_TERMINATED(int P_Terminated, PCB Terminated_Arr[]){
    for(int i=0; i<P_Terminated; i++){
        printf("P%d: ", Terminated_Arr[i].iPID);
        printf("Start: %d, ", Terminated_Arr[i].iStart);
        printf("Finish: %d\n", Terminated_Arr[i].iFinish);
    }
}

void CALCULATE_VAL(int n, PCB P[]){
    double AVR_WAITING = 0;
    double AVR_EXE = 0;
    double AVR_RESPONSE = 0;
    for(int i=0; i<n; i++){
        AVR_RESPONSE += (P[i].iStart - P[i].iArrival)*1.0;
        AVR_WAITING += (P[i].iStart - P[i].iArrival)*1.0;
        AVR_EXE += (P[i].iFinish - P[i].iArrival)*1.0;
    }
    printf("AVR Response: %f\n", AVR_RESPONSE/n);
    printf("AVR Waiting: %f\n", AVR_WAITING/n);
    printf("AVR TaT: %f\n", AVR_EXE/n);
}

void DRAW_GANTT(int Num_Of_Fragment, PCB Gantt[]){
    if(Gantt[0].iStart != 0){
        printf("{0}=");
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    else{
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    for(int i=1; i<Num_Of_Fragment; i++){
        if(i != Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
            else if (Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
        }
        else if (i == Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
            else if(Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
        }
    }
    printf("\n");
```

```
}

void AUTO_INIT_PROCESS(PCB P[], int n){
    srand(time(0));
    for(int i = 0; i < n; i++){
        P[i].iPID = i+1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = rand() % 11 + 2;
    }
}

void SORT_ARRIVAL(int n, PCB P[]){
    for(int j = 0; j < n; j++){
        for(int i = 0; i < n-1; i++){
            if(P[i].iBurst == P[i+1].iBurst && P[i].iArrival > P[i+1].iArrival){
                swap(&P[i], &P[i+1]);
            }
            else{
                continue;
            }
        }
    }

}

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

    int Num_Of_Process, P_Ready = 0, P_Terminated = 0;
    printf("Nhap so luong Process: ");
    scanf("%d", &Num_Of_Process);

    int P_Remain = Num_Of_Process;
    int SELECT = 0;
    printf("Khởi tạo mảng Process tự động [0]\n");
    printf("Khởi tạo thủ công [1]\n");
    printf("Lựa chọn: \n");
    scanf("%d", &SELECT);

    if(SELECT == 1){
        INIT_ARR(Num_Of_Process, Input);
    }
    else{
        AUTO_INIT_PROCESS(Input, Num_Of_Process);
    }

```

```
  //
    printf("------------Input-----------\n");
    PRINT_ARR(Num_Of_Process, Input);
    quickSort_Arrival(Input, 0, Num_Of_Process-1);
  //

    int Time_line = 0;
    PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
    REMOVE_PROCESS(&P_Remain, 0, Input);
    //Cap nhat Time_line luc nay se bang thoi gian Arrival cua Process dau tien nap vao
    Time_line = ReadyQueue[0].iArrival;
    //Khi nao cac chuong trinh chua duoc xu li xong thi thuat toan van chay
    while(P_Terminated < Num_Of_Process){
        //Neu Time_line lon hon Arrival cua cac Process trong Input thi nap no vao ReadyQueue
        while(Time_line >= Input[0].iArrival && P_Remain > 0){
            PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
            REMOVE_PROCESS(&P_Remain, 0, Input);
            quickSort_Burst(ReadyQueue, 0, P_Ready - 1);
            SORT_ARRIVAL(P_Ready, ReadyQueue);    //Việc làm này để đảm bảo sau khi sort các Process có cùng Burst thì cái nào đến trước làm trước
        }
        //Kiem tra neu sau vong nay khong co Process nao trong ReadyQueue nhung Input con thi nap Process vao
        if(P_Ready == 0){
            while(Time_line <= Input[0].iArrival && P_Remain > 0){
                PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
                REMOVE_PROCESS(&P_Remain, 0, Input);
                Time_line = ReadyQueue[P_Remain - 1].iArrival;
                quickSort_Burst(ReadyQueue, 0, P_Ready - 1);
                SORT_ARRIVAL(P_Ready, ReadyQueue);     //Việc làm này để đảm bảo sau khi sort các Process có cùng Burst thì cái nào đến trước làm trước

            }
        }
        //Cap nhat iStart cho Process
        UPDATE_START(P_Terminated, Terminated_Arr, &ReadyQueue[0], &Time_line);
        //Sau vong nay chac chan phai co Process de xu li
        if(P_Ready > 0){
            //Cap nhat iFinish
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;
            //Dua vao Terminated_Arr
```

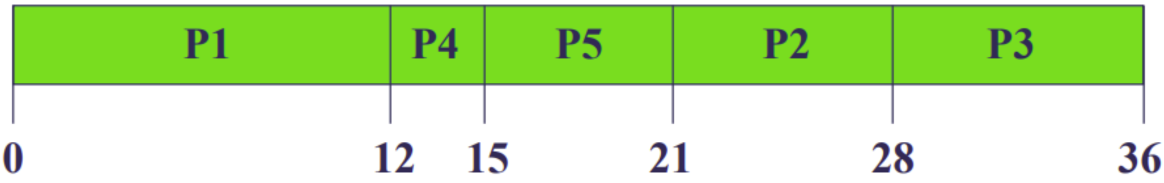
```
                PUSH_PROCESS(&P_Terminated, Terminated_Arr, ReadyQueue[0]);
                //Remove khoi ReadyQueue
                REMOVE_PROCESS(&P_Ready, 0, ReadyQueue);
                //Cap nhat Time_line
                Time_line = Terminated_Arr[P_Terminated - 1].iFinish;
            }
        }

        printf("--------------TERMINATED-----------\n");
        PRINT_TERMINATED(P_Terminated, Terminated_Arr);
        printf("==============GANTT==============\n");
        DRAW_GANTT(P_Terminated, Terminated_Arr);
        printf("--------------AVR------------\n");
        CALCULATE_VAL(P_Terminated, Terminated_Arr);
        return 0;
    }
```

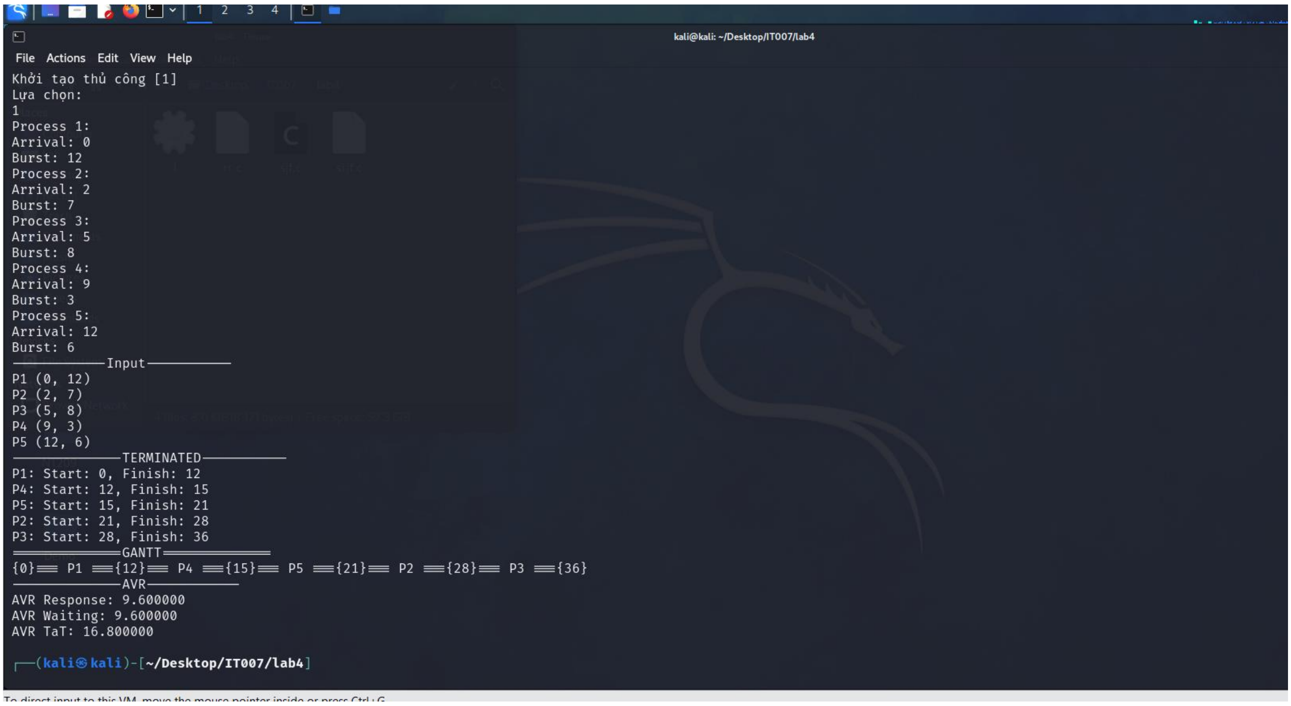
2. Thực thi:

Ví dụ:

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

Giản đồ Gantt





```
Khởi tạo thủ công [1]
Lựa chọn:
1
Process 1:
Arrival: 0
Burst: 12
Process 2:
Arrival: 2
Burst: 7
Process 3:
Arrival: 5
Burst: 8
Process 4:
Arrival: 9
Burst: 3
Process 5:
Arrival: 12
Burst: 6
────────────Input───────────
P1 (0, 12)
P2 (2, 7)
P3 (5, 8)
P4 (9, 3)
P5 (12, 6)
──────────────TERMINATED───────────
P1: Start: 0, Finish: 12
P4: Start: 12, Finish: 15
P5: Start: 15, Finish: 21
P2: Start: 21, Finish: 28
P3: Start: 28, Finish: 36
══════════════GANTT══════════════
{0}══ P1 ══{12}══ P4 ══{15}══ P5 ══{21}══ P2 ══{28}══ P3 ══{36}
──────────────AVR───────────
AVR Response: 9.600000
AVR Waiting: 9.600000
AVR TaT: 16.800000

┌──(kali㉿kali)-[~/Desktop/IT007/lab4]
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G

B – GIẢI THUẬT SRJF

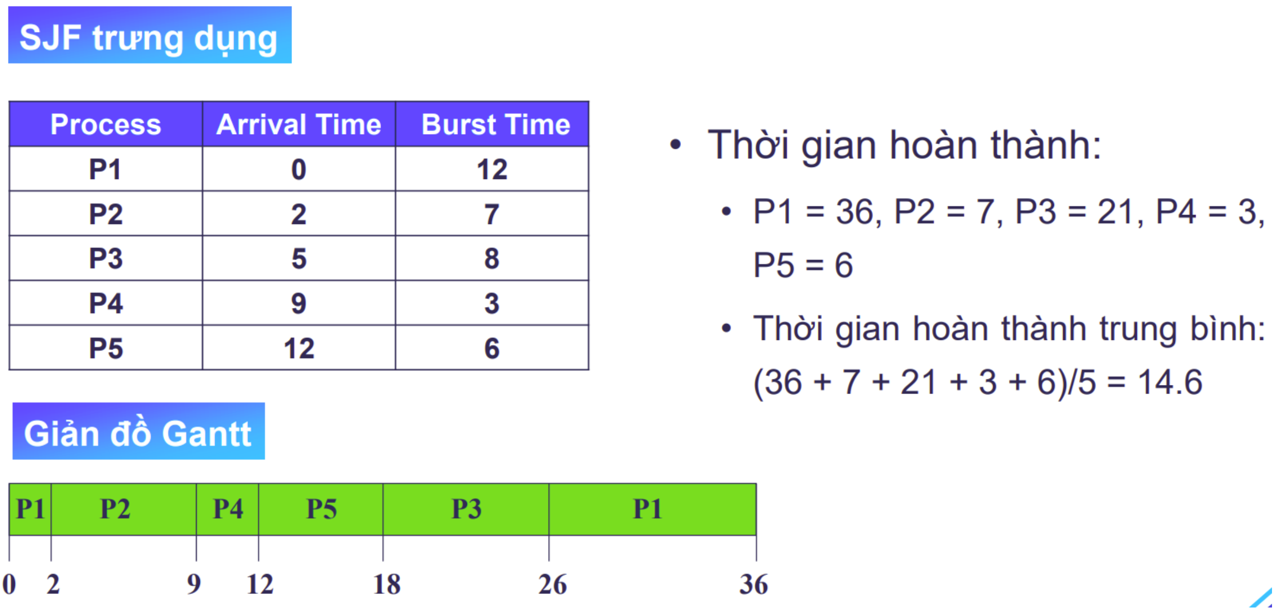
1. Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
typedef struct{
    int iPID, iArrival, iBurst, iBurst_Remain;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
}PCB;

void swap(PCB *A, PCB *B){
    PCB Temp = *A;
    *A = *B;
    *B = Temp;
}

void init_arr(int n, PCB P[]){
    for(int i=0; i<n; i++){
        printf("Process %d: \n", i+1);
        P[i].iPID = i + 1;
        printf("Arrival: ");
        scanf("%d", &P[i].iArrival);
        printf("Burst: ");
        scanf("%d", &P[i].iBurst);
        //==============================
        P[i].iBurst_Remain = P[i].iBurst;
        P[i].iStart = 0;
        P[i].iFinish = 0;
        P[i].iWaiting = 0;
        P[i].iResponse = 0;
        P[i].iTaT = 0;
    }
}

void PRINT_ARR(int n, PCB P[]){
    for(int i=0; i<n && n>0; i++){
        printf("P%d ", P[i].iPID);
        printf("(%d, ", P[i].iArrival);
        printf("%d)\n", P[i].iBurst);
    }
}

void PRINT_TERMINATED(int n, PCB P[]){
    for(int i=0; i<n && n>0; i++){
        printf("P%d: ", P[i].iPID);
        printf("Arrival: %d, ", P[i].iArrival);
        printf("Burst: %d, ", P[i].iBurst);
        printf("Start: %d, ", P[i].iStart);
        printf("Finish: %d, ", P[i].iFinish);
        printf("Response: %d, ", P[i].iResponse);
        printf("Waiting: %d, ", P[i].iWaiting);
        printf("TaT: %d\n", P[i].iTaT);
    }
}

void PRINT_QUEUE(int n, PCB P[]){
    for(int i=0; i<n && n>0; i++){
        printf("P%d: ", P[i].iPID);
        printf("Burst Remain: %d\n", P[i].iBurst_Remain);
    }
}

void PRINT_GANTT_CHART(int n, PCB P[]){
    for(int i=0; i<n && n>0; i++){
        printf("P%d: ", P[i].iPID);
        printf("Start: %d, ", P[i].iStart);
        printf("Finish: %d, ", P[i].iFinish);
        printf("Burst Remain: %d\n", P[i].iBurst_Remain);
    }
}

void PUSH_PROCESS(int *n, PCB P[], PCB Q){
    P[*n] = Q;
    *n = *n + 1;
}

void REMOVE_PROCESS(int *n, int index, PCB P[]){
    for(int i=index; i<*n-1; i++){
        swap(&P[i], &P[i+1]);
    }
    *n = *n - 1;
}

void quickSort_Arrival(PCB a[], int l, int r){
    PCB p=a[(l+r)/2]; ///p: la phan tu privot
    int i=l, j=r;
    while(i<=j){
        while(a[i].iArrival<p.iArrival){
            i++;
        }
        while(a[j].iArrival>p.iArrival){
            j--;
        }
        if(i<=j){
            swap(&a[i],&a[j]);
            i++;
            j--;
        }
    }
    if(i<r){
        quickSort_Arrival(a,i,r);
    }
    if(l<j){
        quickSort_Arrival(a,l,j);
    }
}

void quickSort_BurstRemain(PCB a[], int l, int r){
    PCB p=a[(l+r)/2]; ///p: la phan tu privot
    int i=l, j=r;
    while(i<=j){
        while(a[i].iBurst_Remain<p.iBurst_Remain){
            i++;
        }
        while(a[j].iBurst_Remain>p.iBurst_Remain){
            j--;
        }
        if(i<=j){
            swap(&a[i],&a[j]);
            i++;
            j--;
        }
    }
    if(i<r){
        quickSort_BurstRemain(a,i,r);
    }
    if(l<j){
        quickSort_BurstRemain(a,l,j);
    }
}

void UPDATE_START(int Num_Of_Fragment, PCB Gantt[], PCB *P, int *Time_line){
    if(Num_Of_Fragment == 0){
        P->iStart = P->iArrival;
    }
    else{
        if(P->iArrival <= Gantt[Num_Of_Fragment-1].iFinish){
            P->iStart = Gantt[Num_Of_Fragment-1].iFinish;
        }
        else if(P->iArrival > Gantt[Num_Of_Fragment-1].iFinish){
            P->iStart = P->iArrival;
        }
    }
    *Time_line = P->iStart;
}
```

```
void calculate_form_Gantt_Chart(int Num_of_Fragment, PCB Gantt[], int
Num_of_Process, PCB TerminatedArr[]){
    PCB Calculate_Arr[100];
    for(int i=0; i<Num_of_Process; i++){
        int Start = 0;
        int Frag_of_Process=0;
        int Response=0;
        int Waiting=0;
        int Exe=0;
        //Nhet cac manh cua 1 Process vao 1 arr
        for(int j=0; j<Num_of_Fragment; j++){
            if(Gantt[j].iPID == i+1){
                PUSH_PROCESS(&Frag_of_Process, Calculate_Arr, Gantt[j]);
            }
        }
        //Tinh toan
        for(int j=0; j<Frag_of_Process; j++){
            if(j==0){
                Start = Calculate_Arr[0].iStart;
                Response = Calculate_Arr[0].iStart - Calculate_Arr[0].iArrival;
                Waiting = Calculate_Arr[j].iStart -
Calculate_Arr[j].iArrival;      //Thoi gian doi lan dau
                Exe = Calculate_Arr[Frag_of_Process-1].iFinish -
Calculate_Arr[0].iArrival;
            }
            else{
                Waiting += Calculate_Arr[j].iStart - Calculate_Arr[j-1].iFinish;
            }
        }
        //--------GAN GIA TRI VAO TERMINATED_ARR
        for(int j=0; j<Num_of_Process; j++){
            if(TerminatedArr[j].iPID == i+1){
                TerminatedArr[j].iStart = Start;
                TerminatedArr[j].iResponse = Response;
                TerminatedArr[j].iWaiting = Waiting;
                TerminatedArr[j].iTaT = Exe;
            }
        }
    }
    return;
}

void SORT_ARRIVAL(int n, PCB P[]){
    for(int i=0; i<n; i++){
        for(int j=0; j<n-1; j++)
        if(P[j].iBurst_Remain == P[j+1].iBurst_Remain && P[j].iArrival >
P[j+1].iArrival){
            swap(&P[j], &P[j+1]);
        }
        else continue;
    }
}

void calculate_AVR_Val(int P_Terminated, PCB P[]){
    double AVR_RESPONSE=0;
    double AVR_WAITING=0;
    double AVR_EXECUTION=0;
    for(int i=0; i<P_Terminated; i++){
        AVR_RESPONSE += P[i].iResponse*1.00;
        AVR_WAITING +=P[i].iWaiting*1.00;
        AVR_EXECUTION +=P[i].iTaT*1.00;
    }
    printf("AVR_Response: %f\n", AVR_RESPONSE/P_Terminated);
    printf("AVR_Waiting: %f\n", AVR_WAITING/P_Terminated);
    printf("AVR_TaT: %f\n", AVR_EXECUTION/P_Terminated);
    return;
}

void DRAW_GANTT(int Num_Of_Fragment, PCB Gantt[]){
    if(Gantt[0].iStart != 0){
        printf("{0}=");
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    else{
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    for(int i=1; i<Num_Of_Fragment; i++){
        if(i != Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
            else if (Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
        }
        else if (i == Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
            else if(Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
        }
    }
    printf("\n");
}

void AUTO_INIT_PROCESS(PCB P[], int n){
    srand(time(0));
    for(int i = 0; i < n; i++){
        P[i].iPID = i+1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = rand() % 11 + 2;
        P[i].iBurst_Remain = P[i].iBurst;
    }
}

int main(){
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB Terminated_Arr[10];
    PCB Gantt[50];

    int Num_Of_Process, Num_Of_Fragment = 0, P_Ready = 0, P_Terminated = 0;
    printf("Nhap so luong Process: ");
    scanf("%d", &Num_Of_Process);

    int P_Remain = Num_Of_Process;

    int SELECT = 0;
    printf("Khởi tạo mảng Process tự động [0]\n");
    printf("Khởi tạo thủ công [1]\n");
    printf("Lựa chọn: \n");
    scanf("%d", &SELECT);

    if(SELECT == 1){
        init_arr(Num_Of_Process, Input);
    }
    else{
        AUTO_INIT_PROCESS(Input, Num_Of_Process);
    }

    printf("-------Input---------\n");
    PRINT_ARR(P_Remain, Input);
    quickSort_Arrival(Input, 0, P_Remain-1);
    int Time_line = 0;
    //----------------------------------------
    PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
    REMOVE_PROCESS(&P_Remain, 0, Input);
    Time_line = ReadyQueue[0].iArrival;
    //-----------------------------------------
    while(P_Terminated < Num_Of_Process){
        //Co Interrupt_Flag xay ra khi trong qua trinh thuc thi co Process khac 
chen ngang
        int Interrupt_Flag = 0;
        //Them may cai Process co cung Arrival time vao ReadyQueue
        while(Time_line >= Input[0].iArrival && P_Remain >0){
            PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
            REMOVE_PROCESS(&P_Remain, 0, Input);
            quickSort_BurstRemain(ReadyQueue, 0, P_Ready-1);
            //Sort lai theo arrival de dam bao 2 Process co cung Burst_Remain thi 
Process den truoc duoc uu tien
            SORT_ARRIVAL(P_Ready, ReadyQueue);
        }
        //Neu Time_line do van chua co Process nao den va ReadyQueue dang trong 
thi nap Process vao
        if(Time_line < Input[0].iArrival && P_Ready == 0 && P_Remain > 0){
            PUSH_PROCESS(&P_Ready, ReadyQueue, Input[0]);
            REMOVE_PROCESS(&P_Remain, 0, Input);
        }
        //Cap nhat iStart cho ReadyQueue[0]
        UPDATE_START(Num_Of_Fragment, Gantt, &ReadyQueue[0], &Time_line);
        //Xu li ReadyQueue[0], KIEM TRA trong Input xem co Process nao den khi 
ReadyQueue[0] dang lam viec khong
        for(int i=0; i<P_Remain && P_Remain>0; i++){     //Ham kiem tra nay co 
van de o dau do
            //Co Process den trong khi ReadyQueue[0] chua hoan thanh xong
            if(Input[i].iArrival <= (ReadyQueue[0].iStart + 
ReadyQueue[0].iBurst_Remain)){
                //Kiem tra xem iBurst_Remain cua no co be hon iBurst_Remain cua 
ReadyQueue[0] tai thoi diem no den khong
                if(Input[i].iBurst_Remain < (ReadyQueue[0].iBurst_Remain - 
Input[i].iArrival + ReadyQueue[0].iStart)){
                    //Interrupt xay ra
                    Interrupt_Flag = 1;
                    //Cap nhat Burst_Remain
                    ReadyQueue[0].iBurst_Remain = ReadyQueue[0].iBurst_Remain - 
Input[i].iArrival + ReadyQueue[0].iStart;
                    //Cap nhat Finish
                    ReadyQueue[0].iFinish = Input[i].iArrival;
                    //Cap nhat Time_line
                    Time_line = ReadyQueue[0].iFinish;

                    //KIEM TRA TRUOC KHI DUA VAO GANTT
                    if(Gantt[Num_Of_Fragment-1].iPID != ReadyQueue[0].iPID){
                        //Di chuyen Fragment nay vao Gantt
                        PUSH_PROCESS(&Num_Of_Fragment, Gantt, ReadyQueue[0]);
                    }   //Mat khac chi cap nhat gia tri Finish cho Gantt nay
                    else if(Gantt[Num_Of_Fragment-1].iPID == 
ReadyQueue[0].iPID){
                        Gantt[Num_Of_Fragment-1].iFinish = ReadyQueue[0].iFinish;
                    }

                    //Push Process gay Interrupt nay vao ReadyQueue
                    PUSH_PROCESS(&P_Ready, ReadyQueue, Input[i]);
                    //Xoa Process nay khoi Input
                    REMOVE_PROCESS(&P_Remain, i, Input);
                    //Sort lai ReadyQueue
                    quickSort_BurstRemain(ReadyQueue, 0, P_Ready-1);
                    //Sort lai uu tien Process den truoc neu cung Burst_Remain
                    SORT_ARRIVAL(P_Ready, ReadyQueue);
                    //Cap nhat iSTART cho ReadyQueue[0], cap nhat luon Time_line
                    UPDATE_START(Num_Of_Fragment, Gantt, &ReadyQueue[0], 
&Time_line);
                }
            }
        }
        //Neu co Interrupt_Flag khong bat len tuc la tien doan truoc khong co 
Process nao chen ngang trong qua trinh lam viec thi moi thuc hien xu li buoc nay
        if(Interrupt_Flag == 0){
            //Xu li hoan toan ReayQueue[0]
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart + 
ReadyQueue[0].iBurst_Remain;
            ReadyQueue[0].iBurst_Remain = 0;
            //Nhet ReadyQueue[0] vao Gantt
            PUSH_PROCESS(&Num_Of_Fragment, Gantt, ReadyQueue[0]);
            //Nhet ReadyQueue[0] vao Terminated_Arr
            PUSH_PROCESS(&P_Terminated, Terminated_Arr, ReadyQueue[0]);
            //Xoa ReadyQueue[0] khoi ReadyQueue
            REMOVE_PROCESS(&P_Ready, 0, ReadyQueue);
            //Sort lai ReadyQueue
            quickSort_BurstRemain(ReadyQueue, 0, P_Ready-1);
            //Sort lai theo iArrival
            SORT_ARRIVAL(P_Ready, ReadyQueue);
            //Cap nhat ReadyQueue[0] moi cung nhu Time_line
            UPDATE_START(Num_Of_Fragment, Gantt, &ReadyQueue[0], &Time_line);
        }
    }
    calculate_form_Gantt_Chart(Num_Of_Fragment, Gantt, Num_Of_Fragment, 
Terminated_Arr);
    printf("--------------TERMIATED ARR----------\n");
    PRINT_TERMINATED(P_Terminated, Terminated_Arr);
    printf("--------------GANTT CHART------\n");
    PRINT_GANTT_CHART(Num_Of_Fragment, Gantt);
    //Draw Gantt
    DRAW_GANTT(Num_Of_Fragment, Gantt);
    printf("-----------AVR VAL--------------\n");
    calculate_AVR_Val(P_Terminated, Terminated_Arr);
}
```

2. Thực thi:

Ví dụ:



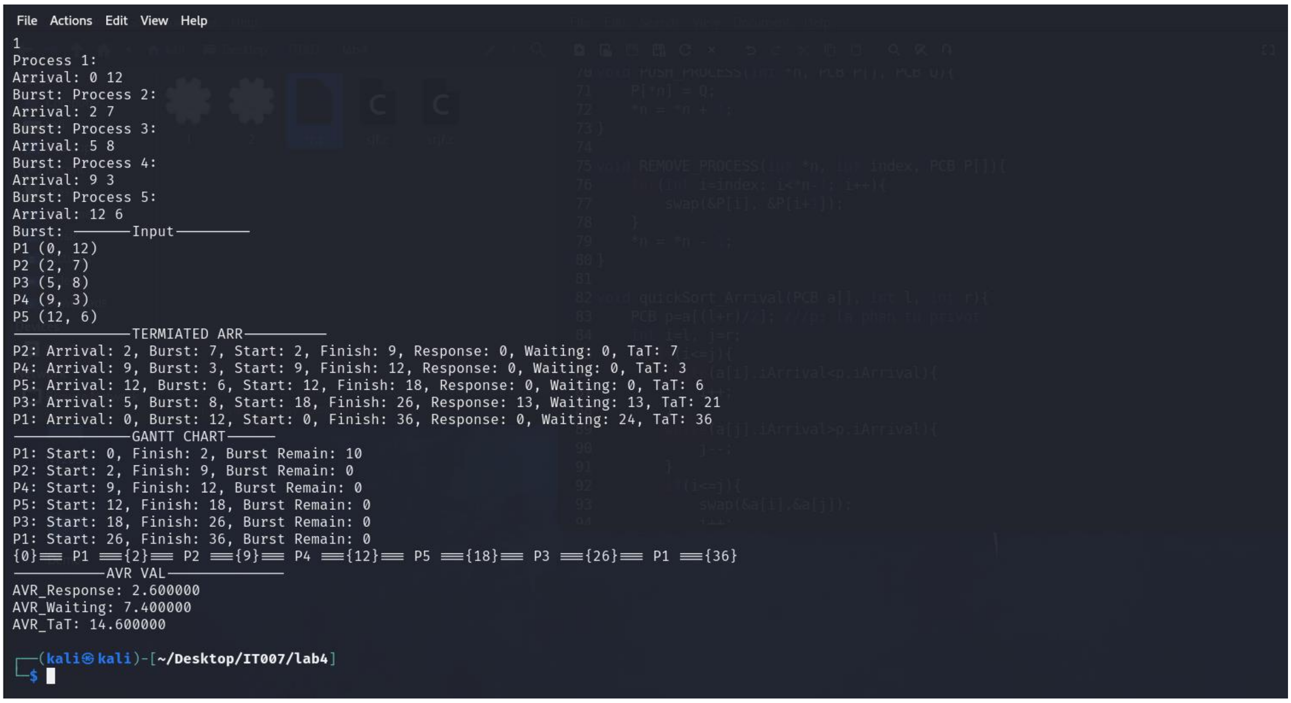
SJF trưng dụng

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

- Thời gian hoàn thành:
  - P1 = 36, P2 = 7, P3 = 21, P4 = 3, P5 = 6
  - Thời gian hoàn thành trung bình: (36 + 7 + 21 + 3 + 6)/5 = 14.6

Giản đồ Gantt

Thực thi chương trình trên Linux:



```
1
Process 1:
Arrival: 0 12
Burst: Process 2:
Arrival: 2 7
Burst: Process 3:
Arrival: 5 8
Burst: Process 4:
Arrival: 9 3
Burst: Process 5:
Arrival: 12 6
Burst: ───────Input─────────
P1 (0, 12)
P2 (2, 7)
P3 (5, 8)
P4 (9, 3)
P5 (12, 6)
──────────────TERMIATED ARR──────────
P2: Arrival: 2, Burst: 7, Start: 2, Finish: 9, Response: 0, Waiting: 0, TaT: 7
P4: Arrival: 9, Burst: 3, Start: 9, Finish: 12, Response: 0, Waiting: 0, TaT: 3
P5: Arrival: 12, Burst: 6, Start: 12, Finish: 18, Response: 0, Waiting: 0, TaT: 6
P3: Arrival: 5, Burst: 8, Start: 18, Finish: 26, Response: 13, Waiting: 13, TaT: 21
P1: Arrival: 0, Burst: 12, Start: 0, Finish: 36, Response: 0, Waiting: 24, TaT: 36
──────────────GANTT CHART──────
P1: Start: 0, Finish: 2, Burst Remain: 10
P2: Start: 2, Finish: 9, Burst Remain: 0
P4: Start: 9, Finish: 12, Burst Remain: 0
P5: Start: 12, Finish: 18, Burst Remain: 0
P3: Start: 18, Finish: 26, Burst Remain: 0
P1: Start: 26, Finish: 36, Burst Remain: 0
{0}══ P1 ══{2}══ P2 ══{9}══ P4 ══{12}══ P5 ══{18}══ P3 ══{26}══ P1 ══{36}
───────────AVR VAL──────────────
AVR_Response: 2.600000
AVR_Waiting: 7.400000
AVR_TaT: 14.600000

┌──(kali㉿kali)-[~/Desktop/IT007/lab4]
└─$ █
```

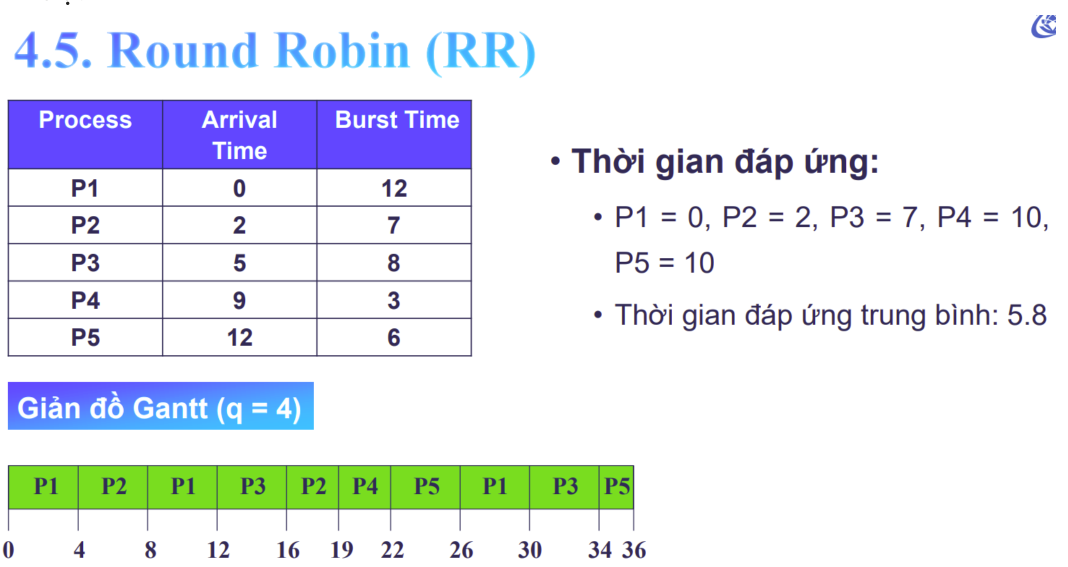
C – GIẢI THUẬT ROUND ROBIN

1. Source code:

```
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
typedef struct {
    int iPID;
    int iArrival;
    int iBurst;
    int iBurst_Remain;
    int iStart;
    int iFinish;
    int iWaiting;
    int iResponse;
    int iTaT;
}PCB;

void swap_Process(PCB *A, PCB *B){
    PCB temp = *A;
    *A = *B;
    *B = temp;
    return;
}

void init_Process_Arr(PCB P[], int n){
    for(int i=0; i<n; i++){
        printf("Process %d: \n", i+1);
        P[i].iPID = i+1;
        printf("Arrival: ");
        scanf("%d", &P[i].iArrival);
        printf("Burst: ");
        scanf("%d", &P[i].iBurst);
        P[i].iBurst_Remain = P[i].iBurst;
    }
    return;
}

void display_Arr(PCB P[], int n){
    for(int i=0; i<n; i++){
        printf("P%d ", P[i].iPID);
        printf("(%d, ", P[i].iArrival);
        printf("%d)\n", P[i].iBurst);
    }
    return;
}

void display_Queue(PCB P[], int n){
    for(int i=0; i<n; i++){
        printf("P%d ", P[i].iPID);
        printf("(%d, ", P[i].iArrival);
        printf("%d)\n", P[i].iBurst_Remain);
    }
    return;
}

void push_Process(int *n, PCB P[], PCB Q){                  //Nạp 1 process vào
hàng đợi, Nạp ở index = 0
    P[*n]=Q;
    *n = *n + 1;
    return;
}

void remove_Process(int *n, int index, PCB P[]){            //Xóa 1 process bất
kì ở ô địa chỉ index và sắp xếp lại mảng
    for(int i=index; i<*n-1; i++){
        swap_Process(&P[i],&P[i+1]);
    }
    *n = *n - 1;
    return;
}

void shellSort(PCB arr[], int n) {                              //Sort theo
iArrival
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i += 1) {
            PCB temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap].iArrival > temp.iArrival; j -=
gap)arr[j] = arr[j - gap];
            arr[j] = temp;
        }
    }
    return;
}

void DISPLAY_TERMINATED_ARR(int P_Terminated, PCB P[]){
    for(int i=0; i<P_Terminated; i++){
        printf("P%d: ", P[i].iPID);
        printf("Start: %d, ", P[i].iStart);
        printf("End: %d, ", P[i].iFinish);
        printf("Respone: %d, ", P[i].iResponse);
        printf("Waiting: %d, ", P[i].iWaiting);
        printf("Execution: %d\n", P[i].iTaT);
    }
    return;
}

void DISPLAY_GANTT_CHART(int Num_Of_Fragment, PCB P[]){
    for(int i=0; i<Num_Of_Fragment; i++){
        printf("P%d: ", P[i].iPID);
        printf("Start: %d, ", P[i].iStart);
        printf("End: %d, ", P[i].iFinish);
        printf("Remain: %d\n", P[i].iBurst_Remain);
    }
    return;
}

void MOVE_TO_END_OF_THE_LINE(int P_Ready, PCB ReadyQueue[]){
    PCB Temp = ReadyQueue[0];
    for(int i=0; i<P_Ready-1; i++){
        swap_Process(&ReadyQueue[i],&ReadyQueue[i+1]);
    }
    ReadyQueue[P_Ready-1]=Temp;
    return;
}
//CHU Y sua ham nay
void UPDATE_DATA_BEFORE_MOVE_TO_GANTT_CHART(PCB *P, int QT, int Num_Of_Fragment,
PCB Gantt_Chart_Arr[]){
    //Nếu Gantt_Chart_Arr chưa có phần tử nào
    if(Num_Of_Fragment == 0){
        P->iStart = P->iArrival;
        if(P->iBurst_Remain > QT){
            P->iFinish = P->iStart + QT;
            P->iBurst_Remain = P->iBurst_Remain - QT;
        }
        else if(P->iBurst_Remain == QT){
            P->iFinish = P->iStart + QT;
            P->iBurst_Remain = 0;
        }
        else if(P->iBurst_Remain < QT){
            P->iFinish = P->iStart + P->iBurst_Remain;
            P->iBurst_Remain = 0;
        }
    }
    //Nếu Gantt_Chart_Arr đã có phần tử
    else if(Num_Of_Fragment>0){
        //START
        if(P->iArrival <= Gantt_Chart_Arr[Num_Of_Fragment-
1].iFinish){         //Trường hợp Process đến trước khi hoàn thành xong
            P->iStart = Gantt_Chart_Arr[Num_Of_Fragment-1].iFinish;
        //BURST_REMAIN, FINISH
            if(P->iBurst_Remain > QT){
                P->iFinish = P->iStart + QT;
                P->iBurst_Remain = P->iBurst_Remain - QT;
            }
            else if(P->iBurst_Remain == QT){
                P->iFinish = P->iStart + QT;
                P->iBurst_Remain = 0;
            }
            else if(P->iBurst_Remain < QT){
                P->iFinish = P->iStart + P->iBurst_Remain;
                P->iBurst_Remain = 0;
            }
        }
        else{                                                   //Trường hợp đến
sau khi hoàn thành xong
        //START
            P->iStart = P->iArrival;
        //BURST_REMAIN, FINISH
            if(P->iBurst_Remain > QT){
                P->iFinish = P->iStart + QT;
                P->iBurst_Remain = P->iBurst_Remain - QT;
            }
            else if(P->iBurst_Remain == QT){
                P->iFinish = P->iStart + QT;
                P->iBurst_Remain = 0;
            }
            else if(P->iBurst_Remain < QT){
                P->iFinish = P->iStart + P->iBurst_Remain;
                P->iBurst_Remain = 0;
            }
        }
    }

    return;
}

void calculate_form_Gantt_Chart(int Num_of_Process, int Num_of_Fragment, PCB
Gantt[], PCB TerminatedArr[]){
    PCB Calculate_Arr[100];
    for(int i=0; i<Num_of_Process; i++){
        int Start = 0;
        int Frag_of_Process=0;
        int Response=0;
        int Waiting=0;
        int Exe=0;
        //Nhet cac manh cua 1 Process vao 1 arr
        for(int j=0; j<Num_of_Fragment; j++){
            if(Gantt[j].iPID == i+1){
                push_Process(&Frag_of_Process, Calculate_Arr, Gantt[j]);
            }
        }
        //Tinh toan
        for(int j=0; j<Frag_of_Process; j++){
            if(j==0){
                Start = Calculate_Arr[0].iStart;
                Response = Calculate_Arr[0].iStart - Calculate_Arr[0].iArrival;
                Waiting = Calculate_Arr[j].iStart - 
Calculate_Arr[j].iArrival;      //Thoi gian doi lan dau
                Exe = Calculate_Arr[Frag_of_Process-1].iFinish - 
Calculate_Arr[0].iArrival;
            }
            else{
                Waiting += Calculate_Arr[j].iStart - Calculate_Arr[j-1].iFinish;
            }
        }
        //--------GAN GIA TRI VAO TERMINATED_ARR
        for(int j=0; j<Num_of_Process; j++){
            if(TerminatedArr[j].iPID == i+1){
                TerminatedArr[j].iStart = Start;
                TerminatedArr[j].iResponse = Response;
                TerminatedArr[j].iWaiting = Waiting;
                TerminatedArr[j].iTaT = Exe;
            }
        }
    }
    return;
}

void calculate_AVR_Val(int P_Terminated, PCB P[]){
    double AVR_RESPONSE=0;
    double AVR_WAITING=0;
    double AVR_EXECUTION=0;
    for(int i=0; i<P_Terminated; i++){
        AVR_RESPONSE += P[i].iResponse*1.00;
        AVR_WAITING +=P[i].iWaiting*1.00;
        AVR_EXECUTION +=P[i].iTaT*1.00;
    }
    printf("AVR_Response: %f\n", AVR_RESPONSE/P_Terminated);
    printf("AVR_Waiting: %f\n", AVR_WAITING/P_Terminated);
    printf("AVR_TaT: %f\n", AVR_EXECUTION/P_Terminated);
    return;
}

void DRAW_GANTT(int Num_Of_Fragment, PCB Gantt[]){
    if(Gantt[0].iStart != 0){
        printf("{0}=");
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    else{
        printf("{%d}=== ", Gantt[0].iStart);
        printf("P%d ===", Gantt[0].iPID);
    }
    for(int i=1; i<Num_Of_Fragment; i++){
        if(i != Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
            else if (Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
            }
        }
        else if (i == Num_Of_Fragment-1){
            if(Gantt[i].iStart == Gantt[i-1].iFinish){
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
            else if(Gantt[i-1].iFinish < Gantt[i].iStart){
                printf("{%d}=", Gantt[i-1].iFinish);
                printf("{%d}=== ", Gantt[i].iStart);
                printf("P%d ===", Gantt[i].iPID);
                printf("{%d}", Gantt[i].iFinish);
            }
        }
    }
    printf("\n");
}

void AUTO_INIT_PROCESS(PCB P[], int n){
    srand(time(0));
    for(int i = 0; i < n; i++){
        P[i].iPID = i+1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = rand() % 11 + 2;
        P[i].iBurst_Remain = P[i].iBurst;
    }
}

int main(){
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB TerminatedArray[10];
    PCB Gantt_Chart[100];

    int QUANTUM_TIME;
    int Num_of_Process, P_Remain;
    int P_Ready=0, P_Termiated=0, Num_Of_Fragment=0;
    do{
        printf("Nhap so luong process (n<=10): ");
        scanf("%d", &Num_of_Process);
    }while(Num_of_Process<=0 || Num_of_Process>10);

    P_Remain = Num_of_Process;
    //====================================
    int SELECT = 0;
    printf("Khởi tạo mảng Process tự động [0]\n");
    printf("Khởi tạo thủ công [1]\n");
    printf("Lựa chọn: \n");
    scanf("%d", &SELECT);
    if(SELECT == 1){
        //Khoi tao thu cong
        init_Process_Arr(Input, Num_of_Process);
    }
    else{
        //Khoi tao tu dong
        AUTO_INIT_PROCESS(Input, Num_of_Process);
    }
    //=====================================
    //Nhap gia tri Quantum Time
    do{
        printf("Nhap Quantum Time: ");
        scanf("%d", &QUANTUM_TIME);
    }while(QUANTUM_TIME<=0);
    //Sort gia tri trong Input
    shellSort(Input, Num_of_Process);                   //Sort lai mang
    //Xuat mang Input ra
    printf("-------------Input-----------\n");
    display_Arr(Input, Num_of_Process);
    //---------------------------------------
    do{
        if(P_Ready==0 && P_Remain>0){
            push_Process(&P_Ready, ReadyQueue, Input[0]);
            remove_Process(&P_Remain, 0, Input);
        }
        //----------------------------------------------
        if(P_Ready>0){
            UPDATE_DATA_BEFORE_MOVE_TO_GANTT_CHART(&ReadyQueue[0], QUANTUM_TIME, 
Num_Of_Fragment, Gantt_Chart);
            //Kiem tra xem Gantt_Chart[Num_Of_Fragment-1] co trung PID voi 
Process duoc them vao khong
            if(Gantt_Chart[Num_Of_Fragment-1].iPID != ReadyQueue[0].iPID){
                push_Process(&Num_Of_Fragment, Gantt_Chart, ReadyQueue[0]);
            }
            else if(Gantt_Chart[Num_Of_Fragment-1].iPID == ReadyQueue[0].iPID){
                Gantt_Chart[Num_Of_Fragment-1].iFinish = ReadyQueue[0].iFinish;
                Gantt_Chart[Num_Of_Fragment-1].iBurst_Remain = 
ReadyQueue[0].iBurst_Remain;
            }
            //Neu ReadyQueue[0].Finish som hon thoi gian den cua cac Process
chuan bi nap vao thi dua cac Process do vao truoc
            while(ReadyQueue[0].iFinish >= Input[0].iArrival && P_Remain>0){
                push_Process(&P_Ready, ReadyQueue, Input[0]);
                remove_Process(&P_Remain, 0, Input);
            }

            if(ReadyQueue[0].iBurst_Remain > 0){
                MOVE_TO_END_OF_THE_LINE(P_Ready, ReadyQueue);
            }
            else{
                push_Process(&P_Termiated, TerminatedArray, ReadyQueue[0]);
                remove_Process(&P_Ready, 0, ReadyQueue);
            }
        }
    }while(P_Termiated < Num_of_Process);
    calculate_form_Gantt_Chart(Num_of_Process, Num_Of_Fragment, Gantt_Chart,
TerminatedArray);
    printf("------TerminatedArr---------\n");
    DISPLAY_TERMINATED_ARR(P_Termiated, TerminatedArray);
    printf("------GANTT CHART-----------\n");
    DISPLAY_GANTT_CHART(Num_Of_Fragment, Gantt_Chart);
    printf("-------------DRAW------------\n");
    DRAW_GANTT(Num_Of_Fragment, Gantt_Chart);
    printf("-----------AVR--------------\n");
    calculate_AVR_Val(P_Termiated, TerminatedArray);
}
```

2. Thực thi:

Ví dụ:



4.5. Round Robin (RR)

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

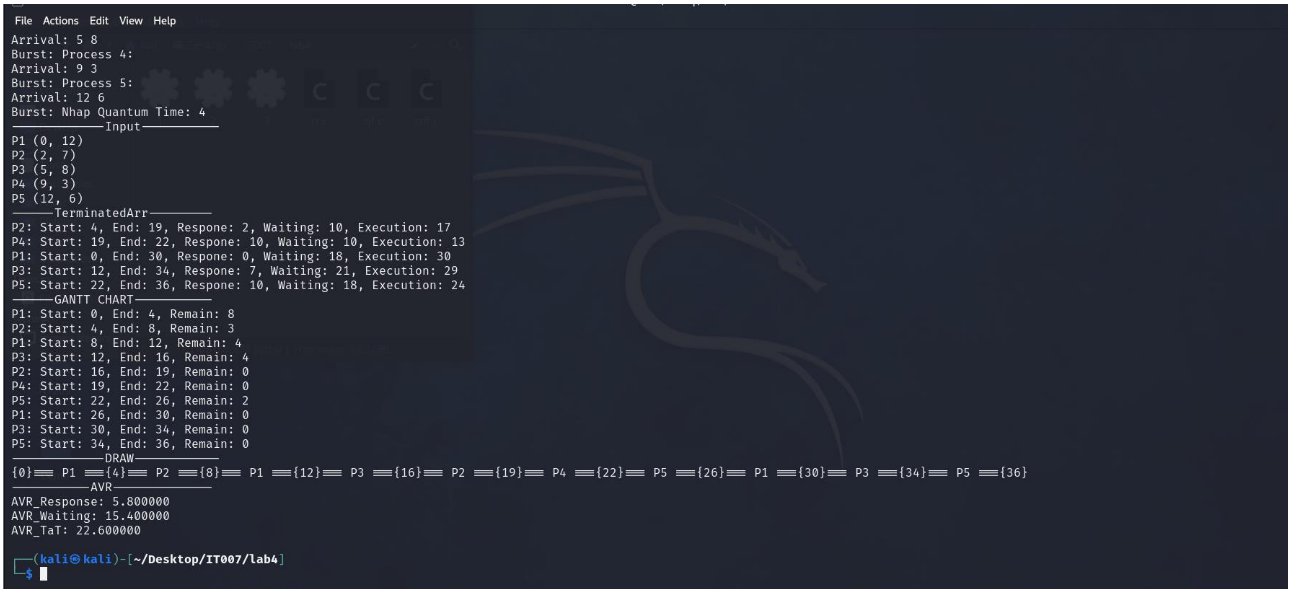
- **Thời gian đáp ứng:**
  - P1 = 0, P2 = 2, P3 = 7, P4 = 10, P5 = 10
  - Thời gian đáp ứng trung bình: 5.8

Giản đồ Gantt (q = 4)

P1	P2	P1	P3	P2	P4	P5	P1	P3	P5

0 4 8 12 16 19 22 26 30 34 36

Thực thi trên linux:



```
Arrival: 5 8
Burst: Process 4:
Arrival: 9 3
Burst: Process 5:
Arrival: 12 6
Burst: Nhap Quantum Time: 4
─────────────Input─────────────
P1 (0, 12)
P2 (2, 7)
P3 (5, 8)
P4 (9, 3)
P5 (12, 6)
──────TerminatedArr─────────
P2: Start: 4, End: 19, Respone: 2, Waiting: 10, Execution: 17
P4: Start: 19, End: 22, Respone: 10, Waiting: 10, Execution: 13
P1: Start: 0, End: 30, Respone: 0, Waiting: 18, Execution: 30
P3: Start: 12, End: 34, Respone: 7, Waiting: 21, Execution: 29
P5: Start: 22, End: 36, Respone: 10, Waiting: 18, Execution: 24
──────GANTT CHART───────────
P1: Start: 0, End: 4, Remain: 8
P2: Start: 4, End: 8, Remain: 3
P1: Start: 8, End: 12, Remain: 4
P3: Start: 12, End: 16, Remain: 4
P2: Start: 16, End: 19, Remain: 0
P4: Start: 19, End: 22, Remain: 0
P5: Start: 22, End: 26, Remain: 2
P1: Start: 26, End: 30, Remain: 0
P3: Start: 30, End: 34, Remain: 0
P5: Start: 34, End: 36, Remain: 0
─────────────DRAW─────────────
{0}══ P1 ══{4}══ P2 ══{8}══ P1 ══{12}══ P3 ══{16}══ P2 ══{19}══ P4 ══{22}══ P5 ══{26}══ P1 ══{30}══ P3 ══{34}══ P5 ══{36}
───────────AVR──────────────
AVR_Response: 5.800000
AVR_Waiting: 15.400000
AVR_TaT: 22.600000

┌──(kali㉿kali)-[~/Desktop/IT007/lab4]
└─$ █
```