**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**A blue logo with a black background

Description automatically generated**

**BÁO CÁO THỰC HÀNH**

**MÔN HỌC: HỆ ĐIỀU HÀNH**

**LAB 4**

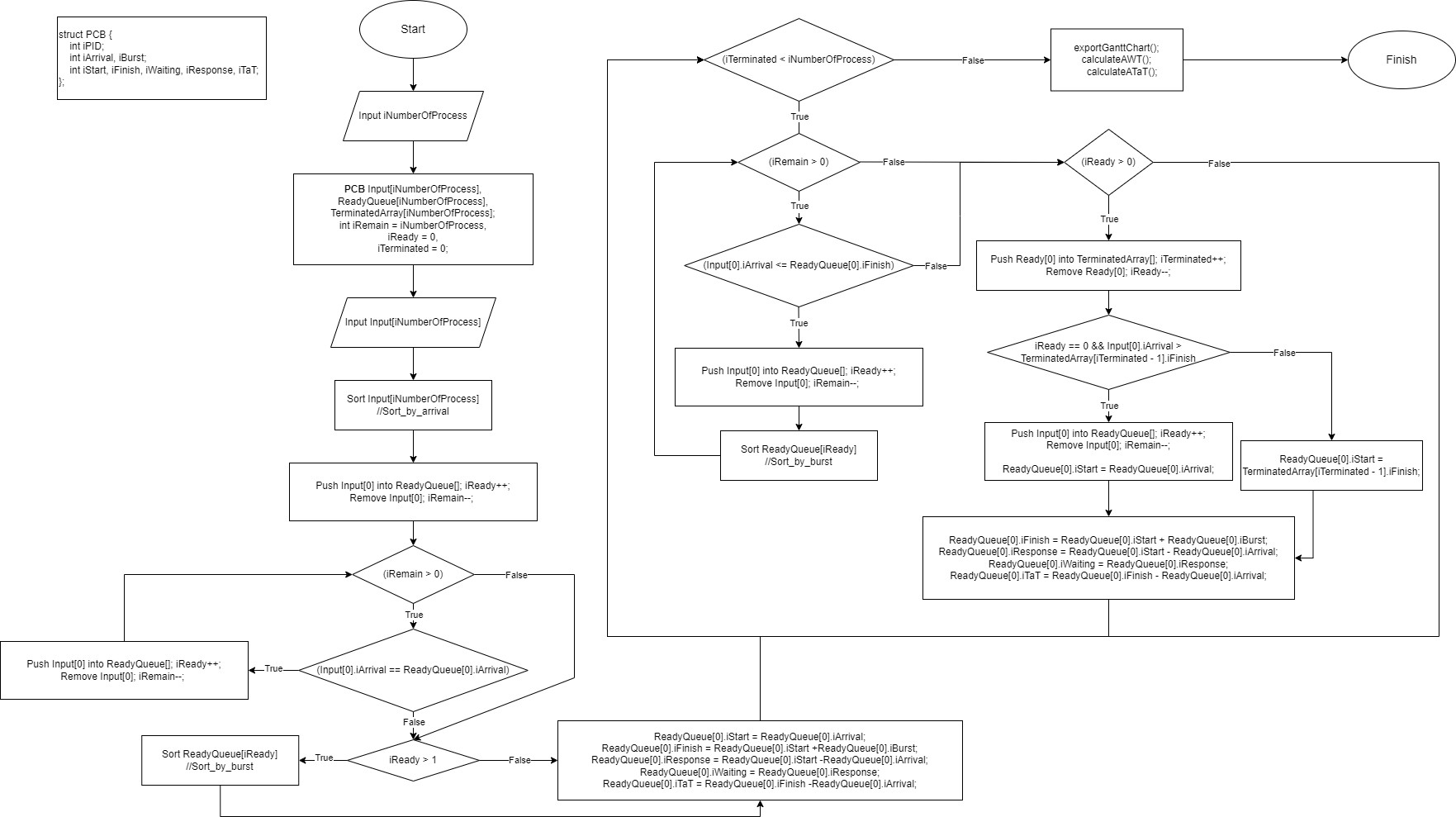
**GVHD : Thầy Lê Hoài Nghĩa**

**Nhóm thực hiện: Nhóm 4**

|  |  |  |
| --- | --- | --- |
| **THÀNH VIÊN NHÓM** | | |
| **Họ và tên** | **MSSV** | **Phân công công việc** |
| Nguyễn Thái Học | 22520488 | Giải thuật SJF : vẽ lưu đồ, chạy tay lưu đồ, thực hiện code, chạy code và kiểm chứng |
| Thạch Minh Hoàng | 22520477 | Giải thuật SRTF : vẽ lưu đồ, chạy tay lưu đồ, thực hiện code, chạy code và kiểm chứng |
| Nguyễn Quốc Hưng | 22520516 | Giải thuật RR : vẽ lưu đồ, chạy tay lưu đồ, thực hiện code, chạy code và kiểm chứng |
| Phạm Nhật Tân | 22521311 | Soạn báo cáo, kiểm chứng lại kết quả chạy tay lưu đồ, code |

# Giải thuật Shortest-Job-First

* + **Lưu đồ:**



*Hình 1. Lưu đồ SJF*

* + **Chạy tay lưu đồ:** iNumberOfProcess = 5 Input[]:

|  |  |  |
| --- | --- | --- |
| STT | Arrival Time | Burst Time |
| 1 | 5 | 2 |
| 2 | 0 | 6 |
| 3 | 9 | 7 |
| 4 | 12 | 8 |
| 5 | 0 | 3 |

iRemain = 5

+ After Sort: Input[]

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

+ Các array ban đầu:

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

|  |  |  |
| --- | --- | --- |
| P4 (9, 7) |  |  |
| P5 (12, 8) |  |  |
| iRemain = 5 | iReady = 0 | iTerminated = 0 |

+ Push Input[0] into ReadyQueue; Remove Input[0] :

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

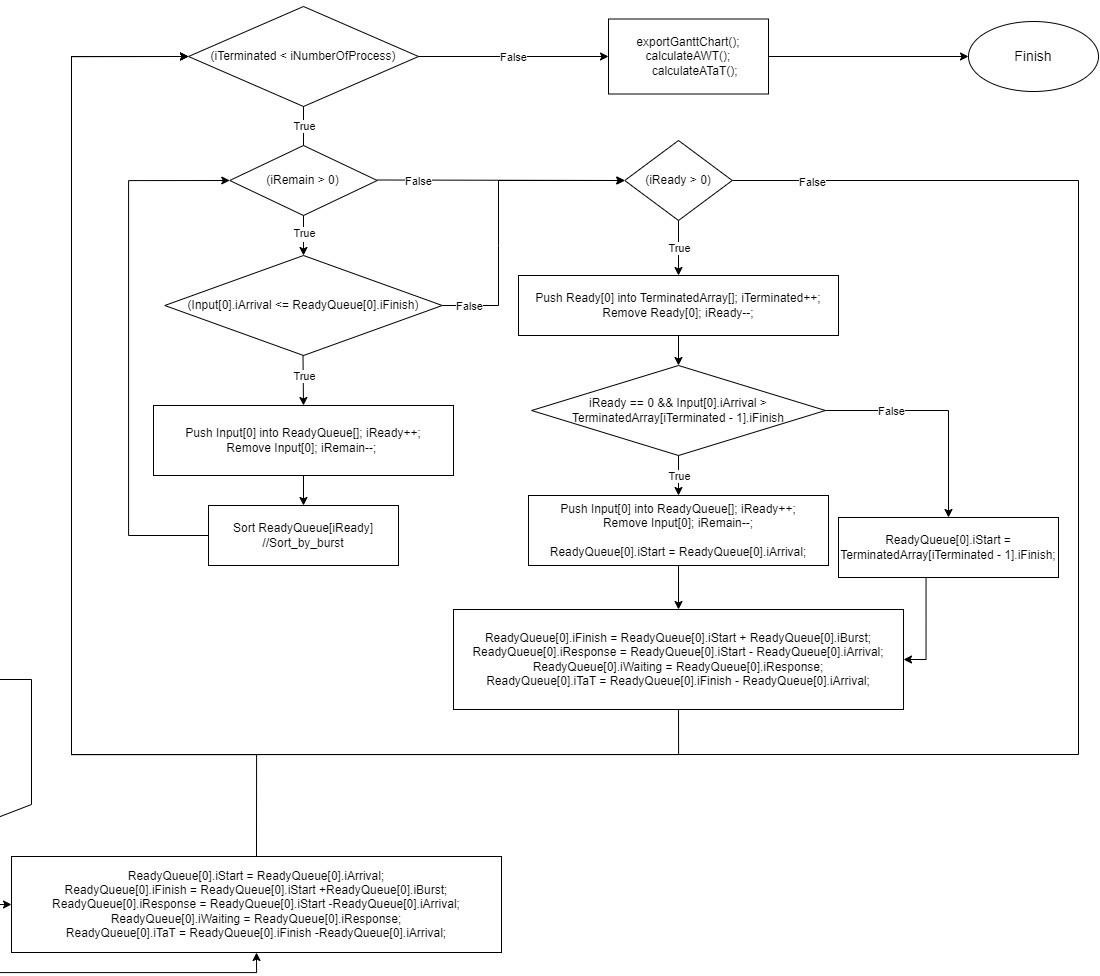
+ Kiểm tra còn tiến trình nào đến cùng thời điểm với P1

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

+ Sort ReadyQueue[]:

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P3 (5, 2) | P2 (0, 3) |  |

|  |  |  |
| --- | --- | --- |
| P4 (9, 7) | P1 (0, 6) |  |
| P5 (12, 8) |  |  |
|  |  |  |
|  |  |  |
| iRemain = 3 | iReady = 2 | iTerminated = 0 |



ReadyQueue[0]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Start | Finish | Response | Waiting | Turnaround |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P2 (0, 3) | 0 | 3 | 0 | 0 | 3 |

P3.iArrival = 5 > 3 => break;

Push ReadyQueue[0] into TerminatedArray[]:

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

ReadyQueue[0]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Start | Finish | Response | Waiting | Turnaround |
| P1 (0, 3) | 3 | 9 | 3 | 3 | 9 |

Làm tương tự với các Process còn lại:

+ Cuối cùng ta sẽ được kết quả như bảng bên dưới:

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

# Code:

#include <stdio.h> #include <stdlib.h> #include <time.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 initPID(int *n*, *PCB P*[])

{

for(int i = 0; i < *n*; i++)

*P*[i].iPID = i + 1;

}

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

{

srand(time(NULL));

for (int i = 0; i < *n*; i++)

{

*P*[i].iArrival = rand() % 21;

*P*[i].iBurst = rand() % 11 + 2;

}

}

void printInput(int *n*, *PCB P*[])

{

printf("\nInput:\n");

printf("+ +\n"); printf("| Process | Arrival | Burst |\n"); printf("+ +\n"); int i;

for (i = 0; i < *n*; i++)

{

printf("| P%-5d | %-5d | %-4d |\n", *P*[i].iPID, *P*[i].iArrival, *P*[i].iBurst);

}

printf("+

+\n");

}

void printProcess(int *n*, *PCB P*[])

{

printf("+ +\n"); printf("| Process | Arrival | Burst | Start | Finish | Turnaround | \n"); printf("+ +\n");

for (int i = 0; i < *n*; i++)

{

printf("| P%-5d | %-5d | %-4d | %-4d | %-4d | %-7d |\n", *P*[i].iPID, *P*[i].iArrival, *P*[i].iBurst, *P*[i].iStart, *P*[i].iFinish, *P*[i].iTaT);

}

printf("+ +\n");

}

void exportGanttChart(int *n*, *PCB P*[])

{

int i; int j;

int countEmpty = 0; for (i = 0; i < *n*; i++)

{

printf("+");

if (*P*[i].iArrival > 0 && i == 0)

{

countEmpty++;

for (j = 0; j <= *P*[0].iArrival; j++) printf("-");

printf("+");

}

for (j = 0; j <= *P*[i].iBurst; j++)

{

printf("-");

if (j == (int)(*P*[i].iBurst/2))

printf("--");

}

if (*P*[i + 1].iArrival > *P*[i].iFinish && i < (*n* - 2))

{

printf("+"); countEmpty++;

for (j = 0; j <= (*P*[i + 1].iArrival - *P*[i].iFinish); j++)

{

printf("-");

}

}

}

printf("+\n");

for (i = 0; i < *n*; i++)

{

printf("|");

if (*P*[i].iArrival > 0 && i == 0)

{

for (j = 0; j <= *P*[0].iArrival; j++) printf(" ");

printf("|");

}

for (j = 0; j <= *P*[i].iBurst; j++)

{

printf(" ");

if (j == (int)(*P*[i].iBurst/2))

printf("P%d", *P*[i].iPID);

}

if (*P*[i + 1].iArrival > *P*[i].iFinish && i < (*n* - 2))

{

printf("|");

for (j = 0; j <= (*P*[i + 1].iArrival - *P*[i].iFinish); j++)

{

printf(" ");

}

}

}

printf("|");

printf("\n");

for (i = 0; i < *n*; i++)

{

printf("+");

if (*P*[0].iArrival > 0 && i == 0)

{

for (j = 0; j <= *P*[0].iArrival; j++) printf("-");

printf("+");

}

for (j = 0; j <= *P*[i].iBurst; j++)

{

printf("-");

if (j == (int)(*P*[i].iBurst/2))

printf("--");

}

if (*P*[i + 1].iArrival > *P*[i].iFinish && i < (*n* - 2))

{

printf("+");

for (j = 0; j <= (*P*[i + 1].iArrival - *P*[i].iFinish); j++)

{

printf("-");

}

}

}

printf("+\n");

printf("0");

for (i = 0; i < *n*; i++)

{

if (*P*[0].iArrival > 0 && i == 0)

{

for (j = 0; j <= *P*[0].iArrival; j++) printf(" ");

printf("%d", *P*[0].iStart);

}

for (j = 0; j <= *P*[i].iBurst; j++)

{

printf(" ");

if (j == (int)(*P*[i].iBurst/2)) printf(" ");

}

//More if(*P*[i].iFinish < 10)

printf(" ");

printf("%d", *P*[i].iFinish);

if (*P*[i + 1].iArrival > *P*[i].iFinish && i < (*n* - 2))

{

for (j = 0; j <= (*P*[i + 1].iArrival - *P*[i].iFinish); j++) printf(" ");

// printf(" ");

printf("%d", *P*[i + 1].iStart);

}

}

if(countEmpty != 0)

{

printf("\n"); printf("\n+ +");

printf("\n| | : No process in ReadyQueue !"); printf("\n+ +");

}

printf("\n");

}

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

{

*P*[\**n*] = *Q*;

\**n* = \**n* + 1;

}

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

{

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

{

*P*[i] = *P*[i + 1];

}

\**n* = \**n* - 1;

}

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

{

*PCB* temp; temp = \**P*;

\**P* = \**Q*;

\**Q* = temp;

}

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

{

int vt = *low* - 1; switch (*iCriteria*)

{

case SORT\_BY\_ARRIVAL:

{

int pivot = *P*[*high*].iArrival; for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iArrival < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

case SORT\_BY\_BURST:

{

int pivot = *P*[*high*].iBurst;

for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iBurst < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

if(*P*[++vt].iBurst != *P*[*high*].iBurst) swapProcess(&*P*[vt], &*P*[*high*]);

return vt;

}

break;

case SORT\_BY\_PID:

{

int pivot = *P*[*high*].iPID;

for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iPID < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

case SORT\_BY\_START:

{

int pivot = *P*[*high*].iStart;

for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iStart < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

default:

break;

}

swapProcess(&*P*[++vt], &*P*[*high*]); return vt;

}

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

{

if (*low* < *high*)

{

int iPivot = partition(*P*, *low*, *high*, *iCriteria*); quickSort(*P*, *low*, iPivot - 1, *iCriteria*); quickSort(*P*, iPivot + 1, *high*, *iCriteria*);

}

}

void calculateAWT(int *n*, *PCB P*[])

{

int sum = 0;

for (int i = 0; i < *n*; i++)

{

sum += *P*[i].iWaiting;

}

printf("\n\t+ AWT = %.2f\n", (double)sum / *n*);

}

void calculateATaT(int *n*, *PCB P*[])

{

int sum = 0;

for (int i = 0; i < *n*; i++)

{

sum += *P*[i].iTaT;

}

printf("\t+ ATaT = %.2f\n\n", (double)sum / *n*);

}

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); initPID(iRemain, Input);

printInput(iNumberOfProcess, Input);

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

// Sua o day

while (iRemain > 0)

{

if(Input[0].iArrival == ReadyQueue[0].iArrival)

{

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

}

else

break;

}

if(iReady > 1)

quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

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;

while (iTerminated < iNumberOfProcess)

{

while (iRemain > 0)

{

if (Input[0].iArrival <= ReadyQueue[0].iFinish)

{

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

quickSort(ReadyQueue, 1, iReady - 1, SORT\_BY\_BURST); continue;

}

else

break;

}

if (iReady > 0)

{

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

1].iFinish)

removeProcess(&iReady, 0, ReadyQueue);

// Xử lý trường hợp không có tiến trình trong ReadyQueue

if (iReady == 0 && Input[0].iArrival > TerminatedArray[iTerminated -

{

pushProcess(&iReady, ReadyQueue, Input[0]); removeProcess(&iRemain, 0, Input); ReadyQueue[0].iStart = ReadyQueue[0].iArrival;

}

else

ReadyQueue[0].iStart = TerminatedArray[iTerminated - 1].iFinish;

// Xử lý trường hợp không có tiến trình trong ReadyQueue

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======================================== SJF Scheduling

========================================\n\n");

printf("Gantt chart:\n"); exportGanttChart(iTerminated, TerminatedArray);

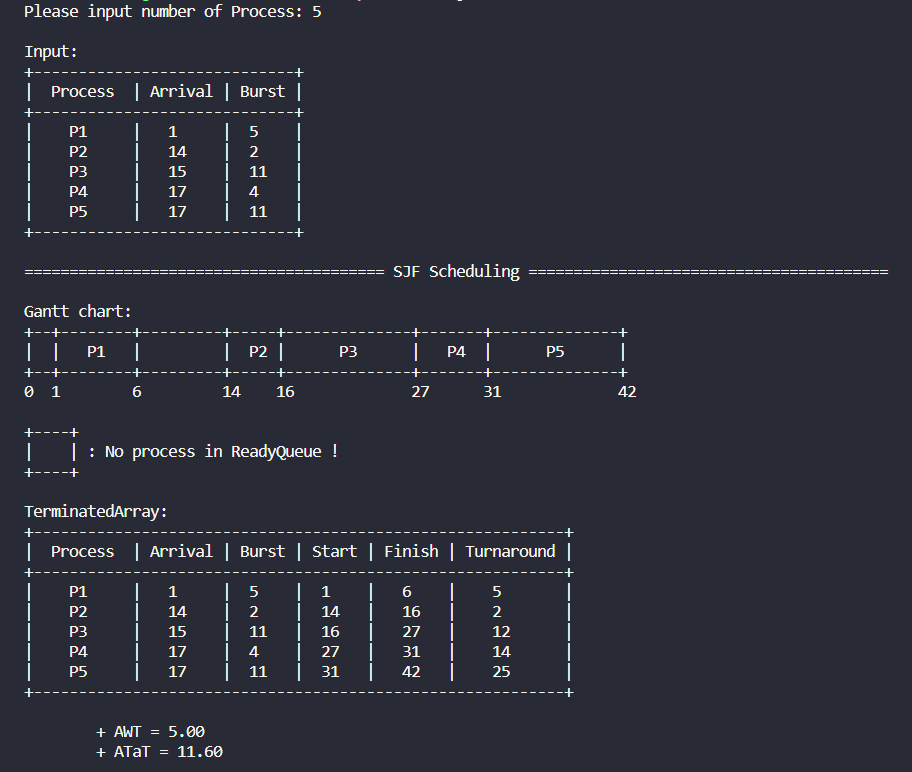
// quickSort(TerminatedArray, 0, iTerminated - 1, SORT\_BY\_PID); printf("\nTerminatedArray:\n");

printProcess(iTerminated, TerminatedArray); calculateAWT(iTerminated, TerminatedArray); calculateATaT(iTerminated, TerminatedArray); return 0;

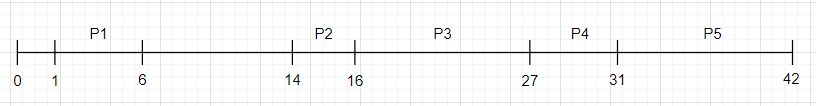
}

* + - Chạy code

+ Test case 1:

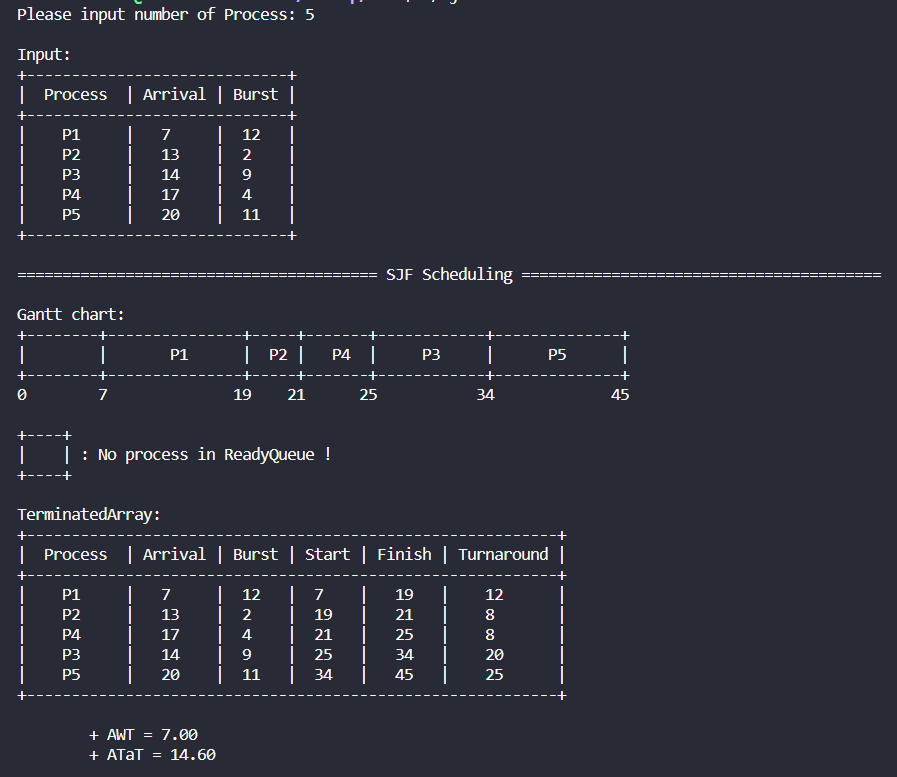


*Hình 2. Test 1- Code*

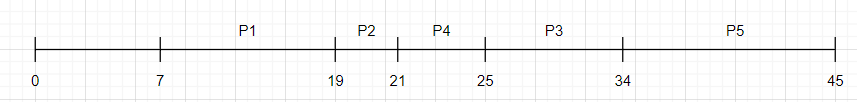


*Hình 3 Test 1- Chạy tay*

+ Test case 2:

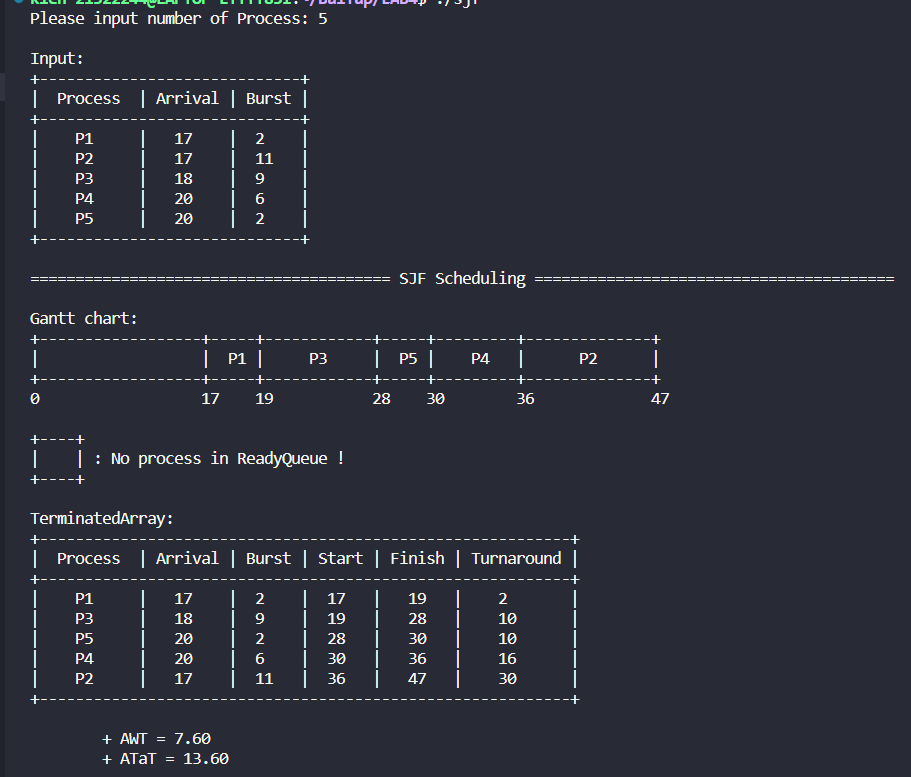


*Hình 3. Test 2- Code*

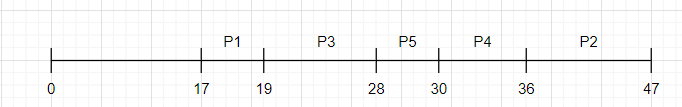


*Hình 4. Test 2- Chạy tay*

+ Test case 3:



*Hình 5. Test 3- Code*

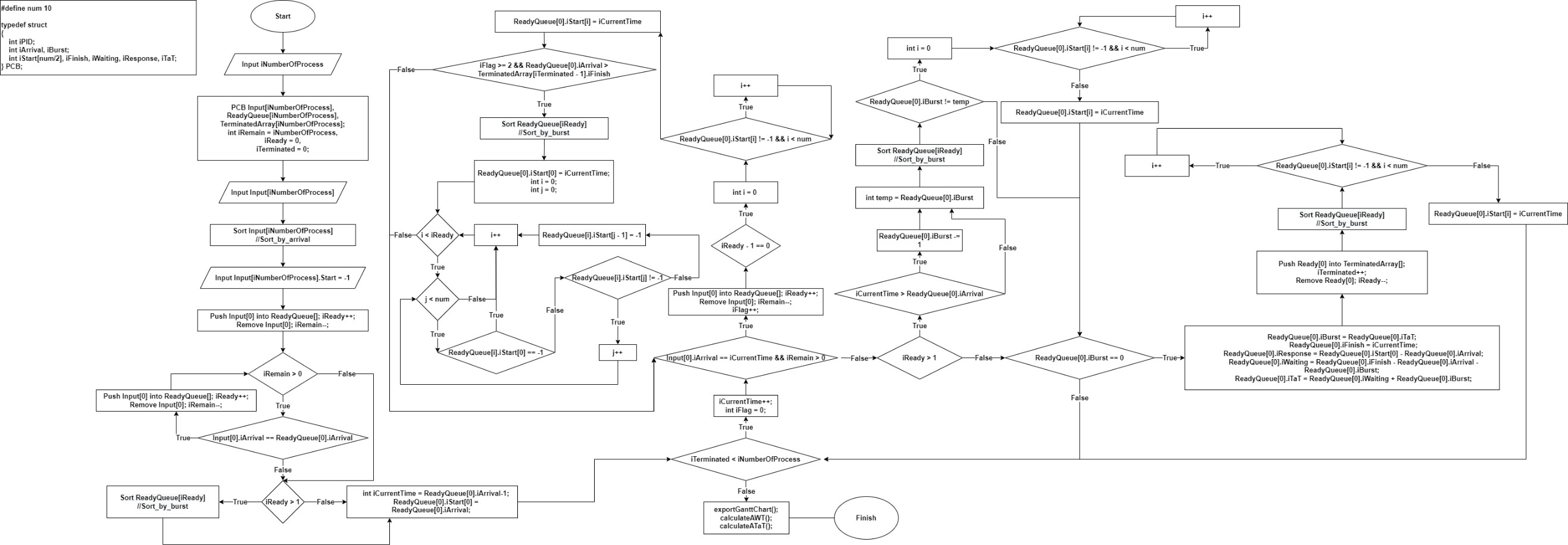


*Hình 6. Test 3- Chạy tay*

# Giải thuật Shortest-Remaining-Time-First hoặc Round Robin

Giải thuật Shortest-Remaining-Time-First (SRTF)

* Lưu đồ giải thuật:



*Hình 4. Lưu đồ SRTF*

* Chạy tay lưu đồ giải thuật:
  + iNumberOfProcess = 0

iRemain = iNumberOfProcess = 0 iReady = 0

iTerminated = 0

* + Input []:

|  |  |  |
| --- | --- | --- |
| Process | Arrival | Burst |
| P1 | 0 | 12 |
| P2 | 5 | 4 |
| P3 | 7 | 6 |
| P4 | 18 | 4 |
| P5 | 20 | 7 |

* + iRemain = 5
  + After Sort:

Input[]:

|  |  |  |
| --- | --- | --- |
| Process | Arrival | Burst |
| P1 | 0 | 12 |
| P2 | 5 | 4 |
| P3 | 7 | 6 |
| P4 | 18 | 4 |
| P5 | 20 | 7 |

* + Input [iNumberOfProcess] = -1
  + Các Array ban đầu:

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P1 (0, 12) |  |  |
| P2 (5, 4) |  |  |
| P3 (7, 6) |  |  |
| P4 (18, 4) |  |  |
| P5 (20, 7) |  |  |

* + Push Input[0] into ReadyQueue[] iReady = 1

Remove Input[0]

iRemain = 4

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P2 (5, 4) | P1 (0, 12) |  |
| P3 (7, 6) |  |  |
| P4 (18, 4) |  |  |
| P5 (20, 7) |  |  |

* + int iCurrentTime = ReadyQueue[0].iArrival – 1 = -1 ReadyQueue[0].iStart[0] = ReadyQueue[0].iArrival

======================

* + iCurrentTime = iCurrentTime + 1 = 0 int iFlag = 0
  + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0:

+ iCurrentTime = ReadyQueue[0].iArrival (bỏ qua bước này)

+ int temp = ReadyQueue[0].iBurst = 12;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P2 (5, 4) | P1 (0, 12) |  |
| P3 (7, 6) |  |  |
| P4 (18, 4) |  |  |
| P5 (20, 7) |  |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst =12 != 0 (Bỏ qua bước này)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị cho tới khi iCurrent = 5 iCurrentTime = iCurrentTime + 1 = 5

int iFlag = 0

* + Input[0].iArrival (=5) == iCurrentTime && iRemain > 0

+ Push Input[0] into ReadyQueue[]

+ iReady = 2

+ Remove Input[0]

+ iRemain = 3

+ iFlag = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P3 (7, 6) | P1 (0, 8) |  |
| P4 (18, 4) | P2 (5, 4) |  |
| P5 (20, 7) |  |  |

+ Do iReady - 1 (= 1) != 0 (bỏ qua bước này)

+ iFlag = 1 (bỏ qua bước này)

* + Do iReady > 0:

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 7

+ int temp = ReadyQueue[0].iBurst = 7;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P3 (7, 6) | P2 (5, 4) |  |
| P4 (18, 4) | P1 (0, 7) |  |
| P5 (20, 7) |  |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst != 0 (bỏ qua bước này)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị cho tới khi iCurrent = 7 iCurrentTime = iCurrentTime + 1 = 7

int iFlag = 0

* + Input[0].iArrival (=7) == iCurrentTime && iRemain > 0

+ Push Input[0] into ReadyQueue[]

+ iReady = 3

+ Remove Input[0]

+ iRemain = 2

+ iFlag = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P2 (5, 3) |  |
| P5 (20, 7) | P1 (0, 7) |  |
|  | P3 (7, 6) |  |

* + Do iReady > 0:

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 2

+ int temp = ReadyQueue[0].iBurst = 2;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P2 (5, 2) |  |
| P5 (20, 7) | P1 (0, 7) |  |
|  | P3 (7, 6) |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst != 0 (bỏ qua bước này)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 9 iCurrentTime = iCurrentTime + 1 = 9

int iFlag = 0

* + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0:

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 0

+ int temp = ReadyQueue[0].iBurst = 0;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P2 (5, 0) |  |
| P5 (20, 7) | P1 (0, 7) |  |
|  | P3 (7, 6) |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst == 0

+ ReadyQueue[0].iBurst = ReadyQueue[0].iTaT;

+ ReadyQueue[0].iFinish = iCurrentTime;

+ ReadyQueue[0].iResponse = ReadyQueue[0].iStart[0] - ReadyQueue[0].iArrival;

+ ReadyQueue[0].iWaiting = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

+ ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst;

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

+ iTerminated = 1

+ removeProcess(&iReady, 0, ReadyQueue);

+ iReady = 2

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P1 (0, 7) | P2 (5, 4) |
| P5 (20, 7) | P3 (7, 6) |  |

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P3 (7, 6) | P2 (5, 4) |
| P5 (20, 7) | P1 (0, 7) |  |

+ ReadyQueue[0].iStart[0] = iCurrentTime = 9 (do ReadyQueue[0].iStart[0] = -1)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 15 iCurrentTime = iCurrentTime + 1 = 15

int iFlag = 0

* + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 0

+ int temp = ReadyQueue[0].iBurst = 0;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P3 (7, 0) | P2 (5, 4) |
| P5 (20, 7) | P1 (0, 7) |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst == 0

+ ReadyQueue[0].iBurst = ReadyQueue[0].iTaT;

+ ReadyQueue[0].iFinish = iCurrentTime;

+ ReadyQueue[0].iResponse = ReadyQueue[0].iStart[0] - ReadyQueue[0].iArrival;

+ ReadyQueue[0].iWaiting = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

+ ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst;

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

+ iTerminated = 2

+ removeProcess(&iReady, 0, ReadyQueue);

+ iReady = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P1 (0, 7) | P2 (5, 4) |
| P5 (20, 7) |  | P3 (7, 6) |

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P4 (18, 4) | P1 (0, 7) | P2 (5, 4) |
| P5 (20, 7) |  | P3 (7, 6) |

+ ReadyQueue[0].iStart[1] = iCurrentTime = 15 (do ReadyQueue[0].iStart[1] != -1, ReadyQueue[0].iStart[0] = 0)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 18 iCurrentTime = iCurrentTime + 1 = 18

int iFlag = 0

* + Input[0].iArrival (=18) == iCurrentTime && iRemain > 0

+ Push Input[0] into ReadyQueue[]

+ iReady = 2

+ Remove Input[0]

+ iRemain = 1

+ iFlag = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P5 (20, 7) | P1 (0, 5) | P2 (5, 4) |
|  | P4 (18, 4) | P3 (7, 6) |

+ Do iReady - 1 (= 1) != 0 (bỏ qua bước này)

+ iFlag = 1 (bỏ qua bước này)

* + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 4

+ int temp = ReadyQueue[0].iBurst = 4;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P5 (20, 7) | P1 (0, 4) | P2 (5, 4) |
|  | P4 (18, 4) | P3 (7, 6) |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst != 0 (bỏ qua bước này)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 20 iCurrentTime = iCurrentTime + 1 = 20

int iFlag = 0

* + Input[0].iArrival (=20) == iCurrentTime && iRemain > 0

+ Push Input[0] into ReadyQueue[]

+ iReady = 3

+ Remove Input[0]

+ iRemain = 0

+ iFlag = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P1 (0, 3) | P2 (5, 4) |
|  | P4 (18, 4) | P3 (7, 6) |
|  | P5 (20, 7) |  |

+ Do iReady - 1 (= 1) != 0 (bỏ qua bước này)

+ iFlag = 1 (bỏ qua bước này)

* + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 2

+ int temp = ReadyQueue[0].iBurst = 2;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P1 (0, 2) | P2 (5, 4) |
|  | P4 (18, 4) | P3 (7, 6) |
|  | P5 (20, 7) |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst != 0 (bỏ qua bước này)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 22 iCurrentTime = iCurrentTime + 1 = 22

int iFlag = 0

* + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 0

+ int temp = ReadyQueue[0].iBurst = 0;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P1 (0, 0) | P2 (5, 4) |
|  | P4 (18, 4) | P3 (7, 6) |
|  | P5 (20, 7) |  |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst == 0

+ ReadyQueue[0].iBurst = ReadyQueue[0].iTaT;

+ ReadyQueue[0].iFinish = iCurrentTime;

+ ReadyQueue[0].iResponse = ReadyQueue[0].iStart[0] - ReadyQueue[0].iArrival;

+ ReadyQueue[0].iWaiting = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

+ ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst;

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

+ iTerminated = 3

+ removeProcess(&iReady, 0, ReadyQueue);

+ iReady = 2

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P4 (18, 4) | P2 (5, 4) |
|  | P5 (20, 7) | P3 (7, 6) |
|  |  | P1 (0, 12) |

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P4 (18, 4) | P2 (5, 4) |
|  | P5 (20, 7) | P3 (7, 6) |
|  |  | P1 (0, 12) |

+ ReadyQueue[0].iStart[0] = iCurrentTime = 22 (do ReadyQueue[0].iStart[0] = -1)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 26

iCurrentTime = iCurrentTime + 1 = 26 int iFlag = 0

* + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 0

+ int temp = ReadyQueue[0].iBurst = 0;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P4 (18, 0) | P2 (5, 4) |
|  | P5 (20, 7) | P3 (7, 6) |
|  |  | P1 (0, 12) |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst == 0

+ ReadyQueue[0].iBurst = ReadyQueue[0].iTaT;

+ ReadyQueue[0].iFinish = iCurrentTime;

+ ReadyQueue[0].iResponse = ReadyQueue[0].iStart[0] - ReadyQueue[0].iArrival;

+ ReadyQueue[0].iWaiting = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

+ ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst;

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

+ iTerminated = 4

+ removeProcess(&iReady, 0, ReadyQueue);

+ iReady = 1

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P5 (20, 7) | P2 (5, 4) |
|  |  | P3 (7, 6) |
|  |  | P1 (0, 12) |
|  |  | P4 (18, 0) |

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P5 (20, 7) | P2 (5, 4) |
|  |  | P3 (7, 6) |
|  |  | P1 (0, 12) |

|  |  |  |
| --- | --- | --- |
|  |  | P4 (18, 0) |

+ ReadyQueue[0].iStart[0] = iCurrentTime = 26 (do ReadyQueue[0].iStart[0] = -1)

======================

* + Giảm ReadyQueue[0].iBurst 1 đơn vị tới khi iCurrent = 33 iCurrentTime = iCurrentTime + 1 = 33

int iFlag = 0

* + Input[0].iArrival != iCurrentTime && iRemain > 0 (bỏ qua bước này)
  + Do iReady > 0

+ iCurrentTime > ReadyQueue[0].iArrival ReadyQueue[0].iBurst = 0

+ int temp = ReadyQueue[0].iBurst = 0;

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST)

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P5 (20, 0) | P2 (5, 4) |
|  |  | P3 (7, 6) |
|  |  | P1 (0, 12) |
|  |  | P4 (18, 0) |

+ ReadyQueue[0].iBurst == temp (bỏ qua bước này)

* + ReadyQueue[0].iBurst == 0

+ ReadyQueue[0].iBurst = ReadyQueue[0].iTaT;

+ ReadyQueue[0].iFinish = iCurrentTime;

+ ReadyQueue[0].iResponse = ReadyQueue[0].iStart[0] - ReadyQueue[0].iArrival;

+ ReadyQueue[0].iWaiting = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

+ ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst;

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

+ iTerminated = 5

+ removeProcess(&iReady, 0, ReadyQueue);

+ iReady = 0

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  |  | P2 (5, 4) |
|  |  | P3 (7, 6) |
|  |  | P1 (0, 12) |
|  |  | P4 (18, 0) |

|  |  |  |
| --- | --- | --- |
|  |  | P5 (20, 7) |

+ quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  |  | P2 (5, 4) |
|  |  | P3 (7, 6) |
|  |  | P1 (0, 12) |
|  |  | P4 (18, 0) |
|  |  | P5 (20, 7) |

Kết quả cuối cùng khi chạy thuật toán với iRemain = 0, iReady = 0, iTerminated = 5

* Code giải thuật

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

#define num 10

#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[num], iFinish, iWaiting, iResponse, iTaT;

} *PCB*;

void initPID(int *n*, *PCB P*[])

{

for(int i = 0; i < *n*; i++)

*P*[i].iPID = i + 1;

}

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

{

srand(time(NULL));

for (int i = 0; i < *n*; i++)

{

*P*[i].iArrival = rand() % 21;

*P*[i].iBurst = rand() % 11 + 2;

}

}

void printInput(int *n*, *PCB P*[])

{

printf("\nInput:\n");

printf("+ +\n"); printf("| Process | Arrival | Burst |\n"); printf("+ +\n"); int i;

for (i = 0; i < *n*; i++)

{

printf("| P%-5d | %-5d | %-4d |\n", *P*[i].iPID, *P*[i].iArrival, *P*[i].iBurst);

}

printf("+

+\n");

}

void printProcess(int *n*, *PCB P*[])

{

printf("+

+\n");

printf("| Process | Arrival | Burst | Start | Finish | Turnaround | Waiting Time|\n");

printf("+

+\n");

for (int i = 0; i < *n*; i++)

{

printf("| P%-5d | %-5d | %-4d | %-4d | %-4d | %-7d | %-5d |\n", *P*[i].iPID, *P*[i].iArrival, *P*[i].iBurst, *P*[i].iStart[0], *P*[i].iFinish, *P*[i].iTaT, *P*[i].iWaiting);

}

printf("+

+\n");

}

int cmpfunc (const void \* *a*, const void \* *b*) { return ( \*(int\*)*a* - \*(int\*)*b* );

}

void exportGanttChart(int *n*, *PCB P*[])

{

//Tao 1 copy cho PCB

*PCB* q[*n*];

for (int i = 0; i < *n*; i++)

{

q[i] = *P*[i];

}

//Tao mang iStartTime va Cho thong so cua mang iStartTime = -1 int iStartTime[*n*\*num];

int iCount = 0;

for (int i = 0; i < *n*\*num; i++)

{

iStartTime[i] = -1;

}

//Gan cac thong so iStart vao mang iStartTime for (int i = 0; i < *n*; i++)

{

for (int j = 0; j < *n*\*num; j++)

{

if (q[i].iStart[j] != -1) iStartTime[iCount++] = q[i].iStart[j];

else

break;

}

}

//Sap xep mang iStartTime

qsort(iStartTime, iCount, sizeof(int), cmpfunc);

//Tren

for (int count = 0; count < iCount; count++)

{

for (int i = 0; i < *n*; i++)

{

for (int j = 0; j < *n*\*num; j++)

{

if (q[i].iStart[j] == iStartTime[count])

{

printf("+ ");

if (q[i].iFinish < iStartTime[count + 1])

{

printf("+ ");

}

}

else if (q[i].iStart[j] == -1 ) break;

else

continue;

}

}

}

printf("+\n");

//Giua

for (int count = 0; count < iCount; count++)

{

for (int i = 0; i < *n*; i++)

{

for (int j = 0; j < *n*\*num; j++)

{

if (q[i].iStart[j] == iStartTime[count])

{

printf("| P%-4d", q[i].iPID);

if (q[i].iFinish < iStartTime[count + 1])

{

printf("| ");

}

}

else if (q[i].iStart[j] == -1 ) break;

else

continue;

}

}

}

printf("|\n");

//Duoi

for (int count = 0; count < iCount; count++)

{

for (int i = 0; i < *n*; i++)

{

for (int j = 0; j < *n*\*num; j++)

{

if (q[i].iStart[j] == iStartTime[count])

{

printf("+ ");

if (q[i].iFinish < iStartTime[count + 1])

{

printf("+ ");

}

}

else if (q[i].iStart[j] == -1 ) break;

else

continue;

}

}

}

printf("+\n");

//Cuoi

int iFinish;

for (int count = 0; count < iCount; count++)

{

for (int i = 0; i < *n*; i++)

{

for (int j = 0; j < *n*\*num; j++)

{

if (q[i].iStart[j] == iStartTime[count])

{

printf("%-8d", iStartTime[count]);

if (q[i].iFinish < iStartTime[count + 1])

{

printf("%-5d", q[i].iFinish);

}

}

else if (q[i].iStart[j] == -1 ) break;

else

continue;

}

iFinish = q[i].iFinish;

}

}

printf("%d", iFinish); printf("\n");

}

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

{

*P*[\**n*] = *Q*;

\**n* = \**n* + 1;

}

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

{

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

{

*P*[i] = *P*[i + 1];

}

\**n* = \**n* - 1;

}

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

{

*PCB* temp; temp = \**P*;

\**P* = \**Q*;

\**Q* = temp;

}

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

{

int vt = *low* - 1; switch (*iCriteria*)

{

case SORT\_BY\_ARRIVAL:

{

int pivot = *P*[*high*].iArrival; for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iArrival <= pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

case SORT\_BY\_BURST:

{

int pivot = *P*[*high*].iBurst;

for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iBurst < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

if(*P*[++vt].iBurst != *P*[*high*].iBurst) swapProcess(&*P*[vt], &*P*[*high*]);

return vt;

}

break;

case SORT\_BY\_PID:

{

int pivot = *P*[*high*].iPID;

for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iPID < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

case SORT\_BY\_START:

{

int pivot = *P*[*high*].iStart[0]; for (int i = *low*; i < *high*; i++)

{

if (*P*[i].iStart[0] < pivot)

{

swapProcess(&*P*[++vt], &*P*[i]);

}

}

}

break;

default:

break;

}

swapProcess(&*P*[++vt], &*P*[*high*]); return vt;

}

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

{

if (*low* < *high*)

{

int iPivot = partition(*P*, *low*, *high*, *iCriteria*); quickSort(*P*, *low*, iPivot - 1, *iCriteria*); quickSort(*P*, iPivot + 1, *high*, *iCriteria*);

}

}

void calculateAWT(int *n*, *PCB P*[])

{

int sum = 0;

for (int i = 0; i < *n*; i++)

{

sum += *P*[i].iWaiting;

}

printf("\n\t+ AWT = %.2f\n", (double)sum / *n*);

}

void calculateATaT(int *n*, *PCB P*[])

{

int sum = 0;

for (int i = 0; i < *n*; i++)

{

sum += *P*[i].iTaT;

}

printf("\t+ ATaT = %.2f\n\n", (double)sum / *n*);

}

int main()

{

*PCB* Input[num];

*PCB* ReadyQueue[num];

*PCB* TerminatedArray[num]; int iNumberOfProcess;

// Input

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); initPID(iRemain, Input);

printInput(iNumberOfProcess, Input);

//Gan gia tri cho start

for (int i = 0; i < iRemain; i++)

{

for (int j= 0; j < num; j++ ) Input[i].iStart[j] = -1;

Input[i].iTaT = Input[i].iBurst;

}

//Cho truong hop nhieu P cung vao ngay ban dau pushProcess(&iReady, ReadyQueue, Input[0]); removeProcess(&iRemain, 0, Input);

while (iRemain > 0)

{

if(Input[0].iArrival == ReadyQueue[0].iArrival)

{

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

}

else

break;

}

if(iReady > 1)

quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

int iCurrentTime = ReadyQueue[0].iArrival-1; ReadyQueue[0].iStart[0] = ReadyQueue[0].iArrival;

while (iTerminated < iNumberOfProcess )

{

iCurrentTime ++; int iFlag = 0;

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

{

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

if(iReady - 1 == 0)

{

for(int i = 0; i < num ; i++)

{

if (ReadyQueue[0].iStart[i] != -1) continue;

else

{

}

}

}

ReadyQueue[0].iStart[i] = iCurrentTime; break;

1].iFinish)

//Xet truong hop nhieu P cung vao sau khi P truoc do hoan thanh if(iFlag >= 2 && ReadyQueue[0].iArrival > TerminatedArray[iTerminated -

{

quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST); ReadyQueue[0].iStart[0] = iCurrentTime;

for (int i = 1; i < iReady; i++)

{

for (int j = 0; j < num ; j++)

{

if (ReadyQueue[i].iStart[0] == -1) break;

if (ReadyQueue[i].iStart[j] != -1) continue;

else

{

ReadyQueue[i].iStart[j - 1] = -1; break;

}

}

}

}

}

if (iReady > 0)

{

if (iCurrentTime > ReadyQueue[0].iArrival) ReadyQueue[0].iBurst -= 1;

int temp = ReadyQueue[0].iBurst; quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

if( ReadyQueue[0].iBurst != temp)

{

for (int i = 0; i < num ; i++)

{

if (ReadyQueue[0].iStart[i] != -1) continue;

else

{

}

}

}

}

ReadyQueue[0].iStart[i] = iCurrentTime; break;

if (ReadyQueue[0].iBurst == 0)

{

ReadyQueue[0].iBurst = ReadyQueue[0].iTaT; ReadyQueue[0].iFinish = iCurrentTime;

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

ReadyQueue[0].iBurst;

ReadyQueue[0].iTaT = ReadyQueue[0].iWaiting + ReadyQueue[0].iBurst; pushProcess(&iTerminated, TerminatedArray, ReadyQueue[0]); removeProcess(&iReady, 0, ReadyQueue);

quickSort(ReadyQueue, 0, iReady - 1, SORT\_BY\_BURST);

for(int i = 0; i < num ; i++)

{

if (ReadyQueue[0].iStart[i] != -1) continue;

else

{

ReadyQueue[0].iStart[i] = iCurrentTime; break;

}

}

}

}

printf("\n============================== SRTF Scheduling

=================================\n\n");

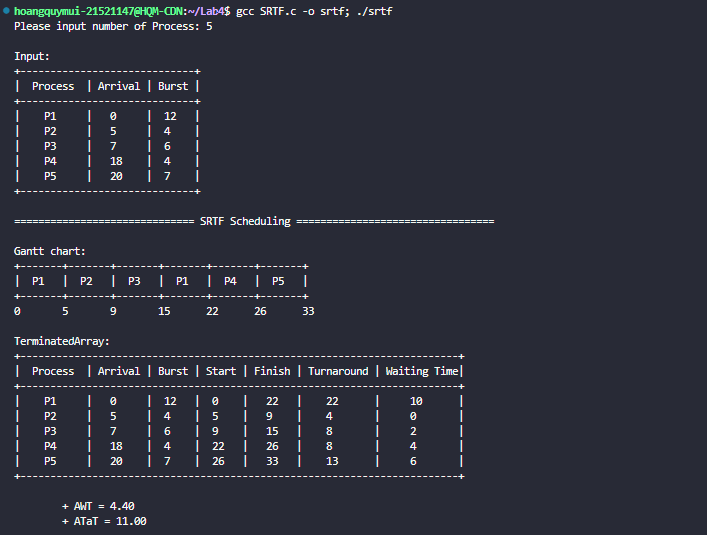
printf("Gantt chart:\n"); exportGanttChart(iTerminated, TerminatedArray); printf("\nTerminatedArray:\n");

quickSort(TerminatedArray, 0, iTerminated - 1, SORT\_BY\_PID); printProcess(iTerminated, TerminatedArray); calculateAWT(iTerminated, TerminatedArray); calculateATaT(iTerminated, TerminatedArray);

return 0;

}

* Trình bày ít nhất 3 testcase
  + Testcase 1



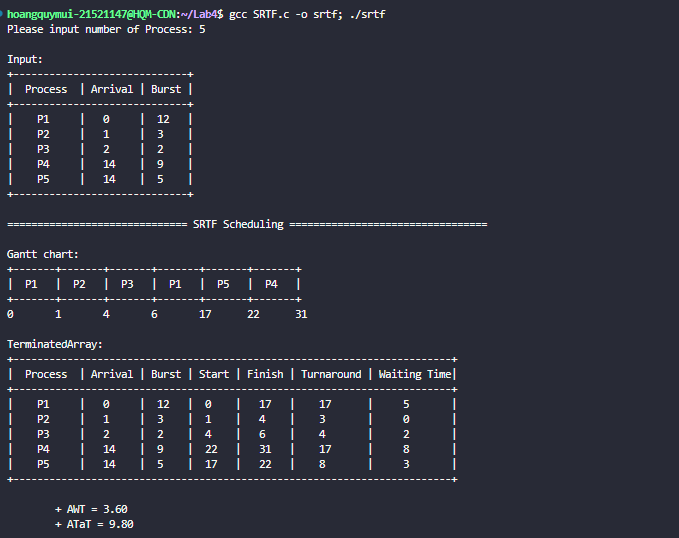
*Hình 7. Test1 1- Code*

A line with numbers and letters

Description automatically generated

* + Testcase 2

*Hình 8. Testcase1 - Chạy tay*



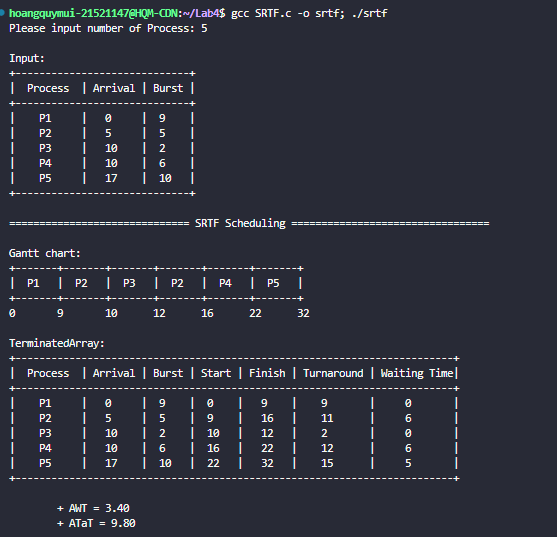
*Hình 9. Testcase 2 - Code*

A line with numbers and letters

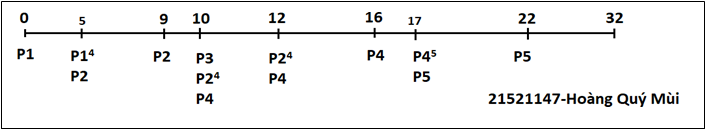
Description automatically generated

* + Testcase 3

*Hình 10. Testcase 2 - Chạy tay*



*Hình 11. Testcase 3 - Code*



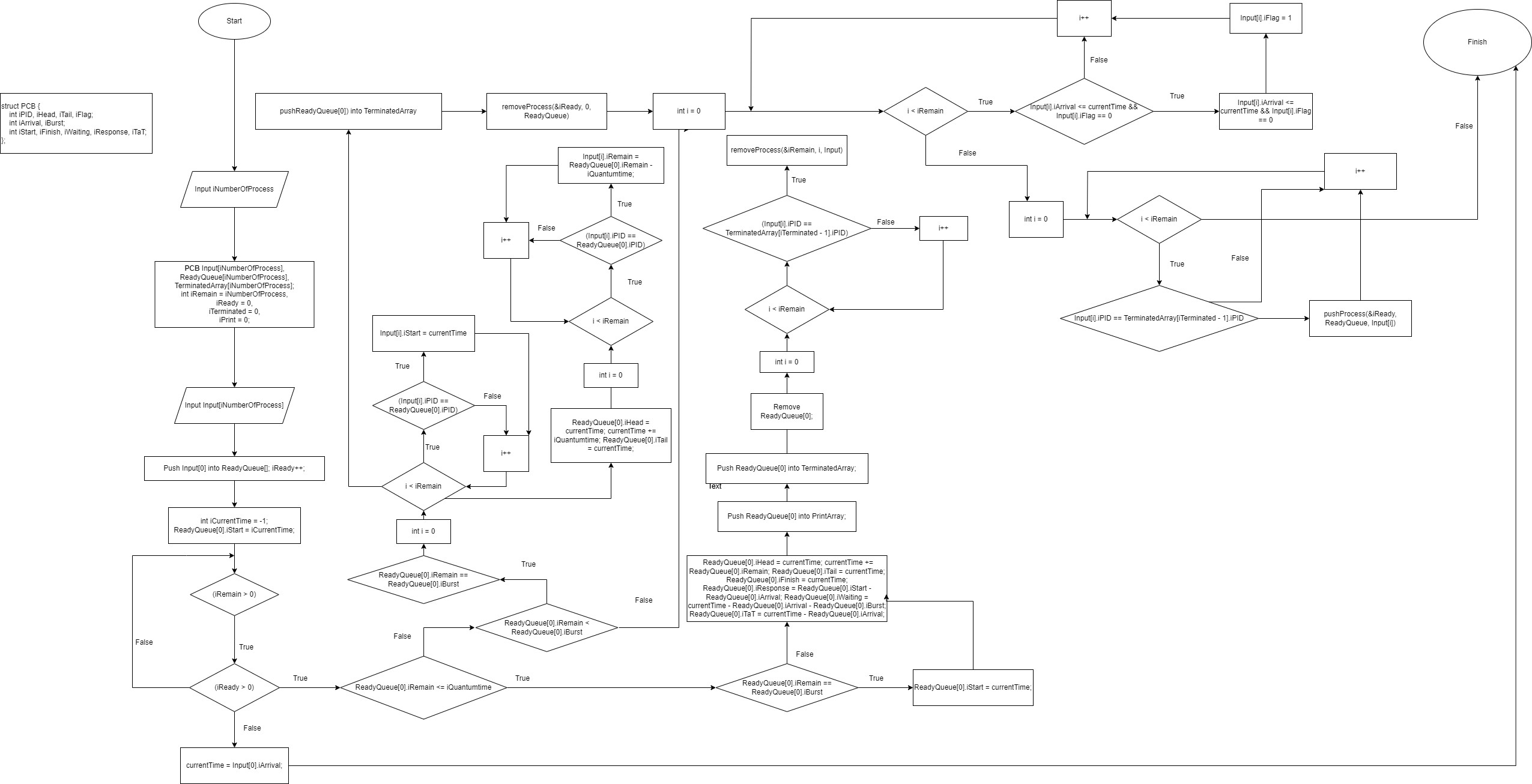
*Hình 12. Testcase 3 - Chạy tay*



# BÀI TẬP ÔN TẬP

**1. Giải thuật Round Robin**

# Lưu đồ



*Hình 5. Lưu đồ RR*

# Chạy tay lưu đồ giải thuật:

iNumberofProcess = 5

iQuantumtime = 0 Input:

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 2 | 11 |
| P2 | 11 | 10 |
| P3 | 17 | 3 |
| P4 | 17 | 5 |
| P5 | 20 | 11 |

Các Array ban đầu

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P1 (2, 11) |  |  |
| P2 (11, 10) |  |  |
| P3 (17, 3) |  |  |
| P4 (17, 5) |  |  |
| P5 (20, 11) |  |  |
| iRemain = 5 | iReady = 0 | iTerminated = 0 |

Set P1 flag to 1 currentTime = 2

P1.iHead = 2

P1.iTail = 7

Push P1 to ReadyQueue

Remove P1 from Input

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P2 (11, 10) | P1 (2, 11) |  |
| P3 (17, 3) |  |  |
| P4 (17, 5) |  |  |
| P5 (20, 11) |  |  |
|  |  |  |
| iRemain = 4 | iReady = 1 | iTerminated = 0 |

Set P1 iFlag currentTime = 7

P1.iHead = 7

P1.iTail = 12

Push P2 to ReadyQueue Remove P2 from Input

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P3 (17, 3) | P2 (11, 10) |  |
| P4 (17, 5) | P1 (2, 11) |  |
| P5 (20, 11) |  |  |
|  |  |  |
|  |  |  |
| iRemain = 4 | iReady = 2 | iTerminated = 0 |

Set P2 iFlag currentTime = 12

P2.iHead = 12

P2.iTail = 17

Push P3 to ReadyQueue Push P4 to ReadyQueue Remove P3 from Input Remove P4 from Input

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P5 (20, 11) | P1 (2, 11) |  |
|  | P3 (17, 3) |  |
|  | P4 (17, 5) |  |
|  | P2 (11, 10) |  |
|  |  |  |
| iRemain = 1 | iReady = 4 | iTerminated = 0 |

Set P1 iFlag P1.iHead = 17

P1.iTail = 18

Push P1 to TerminatedArray Remove P1 to ReadyQueue

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
| P5 (20, 11) | P3 (17, 3) | P1 (2, 11) |
|  | P4 (17, 5) |  |
|  | P2 (11, 10) |  |
|  |  |  |
|  |  |  |
| iRemain = 1 | iReady = 4 | iTerminated = 1 |

Set P3 iFlag P3.iHead = 18

P3.iTail = 21

Push P3 to TerminatedArray Push P5 to ReadyQueue Remove P3 from ReadyQueue Remove P5 from Input

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P4 (17, 5) | P1 (2, 11) |
|  | P2 (11, 10) | P3 (17, 3) |
|  | P5 (20, 11) |  |
|  |  |  |
|  |  |  |
| iRemain = 0 | iReady = 3 | iTerminated = 2 |

Set P4 iFlag P4.iHead = 21

P4.iTail = 26

Push P4 to TerminatedArray Remove P4 from ReadyQueue

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P2 (11, 10) | P1 (2, 11) |
|  | P5 (20, 11) | P3 (17, 3) |

|  |  |  |
| --- | --- | --- |
|  |  | P4 (17, 5) |
|  |  |  |
|  |  |  |
| iRemain = 0 | iReady = 2 | iTerminated = 2 |

Set P2 iFlag P2.iHead = 26

P2.iTail = 31

Push P2 to TerminatedArray Remove P2 from ReadyQueue

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  | P5 (20, 11) | P1 (2, 11) |
|  |  | P3 (17, 3) |
|  |  | P4 (17, 5) |
|  |  | P2 (11, 10) |
|  |  |  |
| iRemain = 0 | iReady = 1 | iTerminated = 2 |

Còn lại P5 Set P5 iFlag

P5.iHead = 31

P5.iTail = 42

Push P5 to TerminatedArray Remove P5 from ReadyQueue

|  |  |  |
| --- | --- | --- |
| Input[] | ReadyQueue[] | TerminatedArray[] |
|  |  | P1 (2, 11) |

|  |  |  |
| --- | --- | --- |
|  |  | P3 (17, 3) |
|  |  | P4 (17, 5) |
|  |  | P2 (11, 10) |
|  |  | P5 (20, 11) |
| iRemain = 0 | iReady = 0 | iTerminated = 5 |

# Code

#include <stdio.h> #include <stdlib.h> #include <time.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, iHead, iTail, iFlag; int iArrival, iBurst, iRemain;

int iStart, iFinish, iWaiting, iResponse, iTaT;

} PCB;

void inputProcess(int n, PCB P[]); void printInput(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[]); void 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[]);

void inputProcess(int n, PCB P[])

{

srand(time(NULL)); PCB Arr[30], Bur[30];

for (int i = 0; i < n; i++)

{

Arr[i].iArrival = rand() % 21; Bur[i].iBurst = rand() % 11 + 2;

}

quickSort(Arr, 0, n - 1, SORT\_BY\_ARRIVAL);

quickSort(Bur, 0, n - 1, SORT\_BY\_BURST); for (int i = 0; i < n; i++)

{

P[i].iArrival = Arr[i].iArrival; P[i].iBurst = Bur[i].iBurst; P[i].iRemain = P[i].iBurst; P[i].iPID = i + 1;

P[i].iFlag = 0;

}

}

void printInput(int n, PCB P[])

{

printf("\nInput:\n");

printf("+ +\n"); printf("| Process | Arrival | Burst |\n"); printf("+ +\n"); int i;

for (i = 0; i < n; i++)

{

printf("| P%-5d | %-5d | %-4d |\n", P[i].iPID, P[i].iArrival, P[i].iBurst);

}

printf("+ +\n");

}

void printProcess(int n, PCB P[])

{

printf("+ +\n"); printf("| Process | Arrival | Burst | Start | Finish | Turnaround | \n"); printf("+ +\n");

for (int i = 0; i < n; i++)

{

printf("| P%-5d | %-5d | %-4d | %-4d | %-4d | %-7d |\n", P[i].iPID, P[i].iArrival, P[i].iBurst, P[i].iStart, P[i].iFinish, P[i].iTaT);

}

printf("+ +\n");

}

void exportGanttChart(int n, PCB P[])

{

int i; int j;

for (i = 0; i < n; i++)

{

printf(" ");

if (P[i].iHead > 0 && i == 0)

{

for (j = 0; j <= P[0].iHead; j++) printf("-");

printf(" ");

}

for (j = 0; j <= P[i].iTail - P[i].iHead; j++)

{

printf("-");

if (j == (int)((P[i].iTail - P[i].iHead) / 2)) printf("--");

}

if (P[i + 1].iHead > P[i].iTail && i < (n - 2))

{

printf(" ");

for (j = 0; j <= (P[i + 1].iHead - P[i].iTail); j++) printf("-");

}

}

printf("\n");

for (i = 0; i < n; i++)

{

printf("|");

if (P[i].iHead > 0 && i == 0)

{

for (j = 0; j <= P[0].iHead; j++) printf(" ");

printf("|");

}

for (j = 0; j <= P[i].iTail - P[i].iHead; j++)

{

printf(" ");

if (j == (int)((P[i].iTail - P[i].iHead) / 2)) printf("P%d", P[i].iPID);

}

if (P[i + 1].iHead > P[i].iTail && i < (n - 2))

{

printf("|");

for (j = 0; j <= (P[i + 1].iHead - P[i].iTail); j++) printf(" ");

}

}

printf("|");

printf("\n");

for (i = 0; i < n; i++)

{

printf(" ");

if (P[0].iHead > 0 && i == 0)

{

for (j = 0; j <= P[0].iHead; j++) printf("-");

printf(" ");

}

for (j = 0; j <= P[i].iTail - P[i].iHead; j++)

{

printf("-");

if (j == (int)((P[i].iTail - P[i].iHead) / 2)) printf("--");

}

if (P[i + 1].iHead > P[i].iTail && i < (n - 2))

{

printf(" ");

for (j = 0; j <= (P[i + 1].iHead - P[i].iTail); j++) printf("-");

}

}

printf("\n");

printf("0");

for (i = 0; i < n; i++)

{

if (P[0].iHead > 0 && i == 0)

{

for (j = 0; j <= P[0].iHead; j++) printf(" ");

printf("%d", P[0].iHead);

}

for (j = 0; j <= P[i].iTail - P[i].iHead; j++)

{

printf(" ");

if (j == (int)((P[i].iTail - P[i].iHead) / 2)) printf(" ");

}

printf("%2d", P[i].iTail);

if (P[i + 1].iHead > P[i].iTail && i < (n - 2))

{

for (j = 0; j <= (P[i + 1].iHead - P[i].iTail); j++) printf(" ");

printf("%2d", P[i + 1].iHead);

}

}

printf("\n");

}

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 = \*n - 1;

}

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

{

PCB temp; temp = \*P;

\*P = \*Q;

\*Q = temp;

}

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

{

PCB \*pivot = &P[high]; int left = low;

int right = high - 1; if (iCriteria == 0)

{

while (1)

{

while (left <= right && P[left].iArrival < pivot->iArrival) left++;

while (right >= left && P[right].iArrival > pivot->iArrival) right--;

if (left >= right) break;

swapProcess(&P[left], &P[right]); left++;

right--;

}

swapProcess(&P[left], &P[high]); return left;

}

if (iCriteria == 1)

{

while (1)

{

while (left <= right && P[left].iPID < pivot->iPID) left++;

while (right >= left && P[right].iPID > pivot->iPID) right--;

if (left >= right) break;

swapProcess(&P[left], &P[right]); left++;

right--;

}

swapProcess(&P[left], &P[high]); return left;

}

}

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("\nAWT = %.2f\n", (double)sum / n);

}

void calculateATaT(int n, PCB P[])

{

int sum = 0;

for (int i = 0; i < n; i++)

{

sum += P[i].iTaT;

}

printf("ATaT = %.2f\n", (double)sum / n);

}

int main()

{

PCB Input[20];

PCB ReadyQueue[20];

PCB TerminatedArray[20]; PCB PrintArray[20];

int iNumberOfProcess; int iQuantumtime;

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

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

inputProcess(iNumberOfProcess, Input); printInput(iNumberOfProcess, Input);

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

Input[0].iFlag = 1;

int currentTime = Input[0].iArrival;

while (iRemain > 0)

{

if (iReady > 0)

{

if (ReadyQueue[0].iRemain <= iQuantumtime)

{

if (ReadyQueue[0].iRemain == ReadyQueue[0].iBurst) ReadyQueue[0].iStart = currentTime;

ReadyQueue[0].iHead = currentTime; currentTime += ReadyQueue[0].iRemain; ReadyQueue[0].iTail = currentTime; ReadyQueue[0].iFinish = currentTime;

ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival; ReadyQueue[0].iWaiting = currentTime - ReadyQueue[0].iArrival -

ReadyQueue[0].iBurst;

ReadyQueue[0].iTaT = currentTime - ReadyQueue[0].iArrival;

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

/\* Xóa tiến trình đã hoàn thành khỏi mảng \*/

for (int i = 0; i < iRemain; i++)

{

if (Input[i].iPID == TerminatedArray[iTerminated - 1].iPID) removeProcess(&iRemain, i, Input);

}

}

else if (ReadyQueue[0].iRemain > iQuantumtime)

{

if (ReadyQueue[0].iRemain == ReadyQueue[0].iBurst)

{

for (int i = 0; i < iRemain; i++)

{

if (Input[i].iPID == ReadyQueue[0].iPID) Input[i].iStart = currentTime;

}

}

ReadyQueue[0].iHead = currentTime;

currentTime += iQuantumtime; ReadyQueue[0].iTail = currentTime;

/\* Cập nhật thời gian còn lại cho tiến trình đang được thực hiện \*/

for (int i = 0; i < iRemain; i++)

{

if (Input[i].iPID == ReadyQueue[0].iPID)

Input[i].iRemain = ReadyQueue[0].iRemain - iQuantumtime;

}

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

}

chưa \*/

/\* Duyệt qua tất cả các tiến trình và kiểm tra xem đã có tiến trình mới đến

for (int i = 0; i < iRemain; i++)

{

if (Input[i].iArrival <= currentTime && Input[i].iFlag == 0)

{

pushProcess(&iReady, ReadyQueue, Input[i]); Input[i].iFlag = 1;

}

}

/\* Duyệt qua tất cả các tiến trình đã hoàn thành và đưa vào hàng đợi \*/

for (int i = 0; i < iRemain; i++)

{

if (Input[i].iPID == TerminatedArray[iTerminated - 1].iPID) pushProcess(&iReady, ReadyQueue, Input[i]);

}

}

else

{

/\* Nếu chưa có tiến trình nào đợi \*/

currentTime = Input[0].iArrival;

}

}

printf("\n==================== Round Robin Scheduling ====================\n\n"); printf("\nGantt Chart:\n");

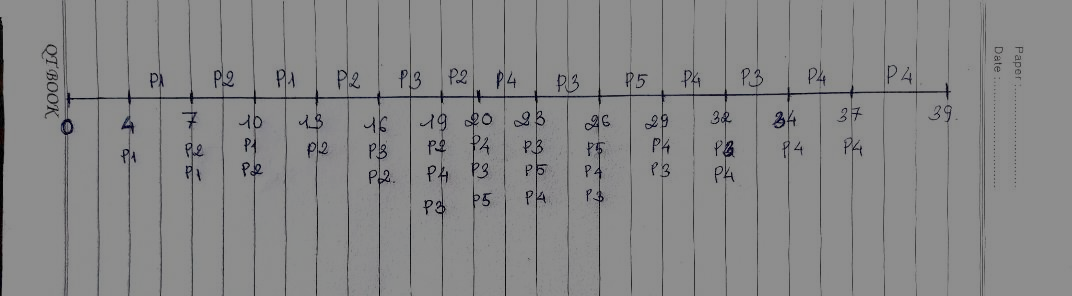
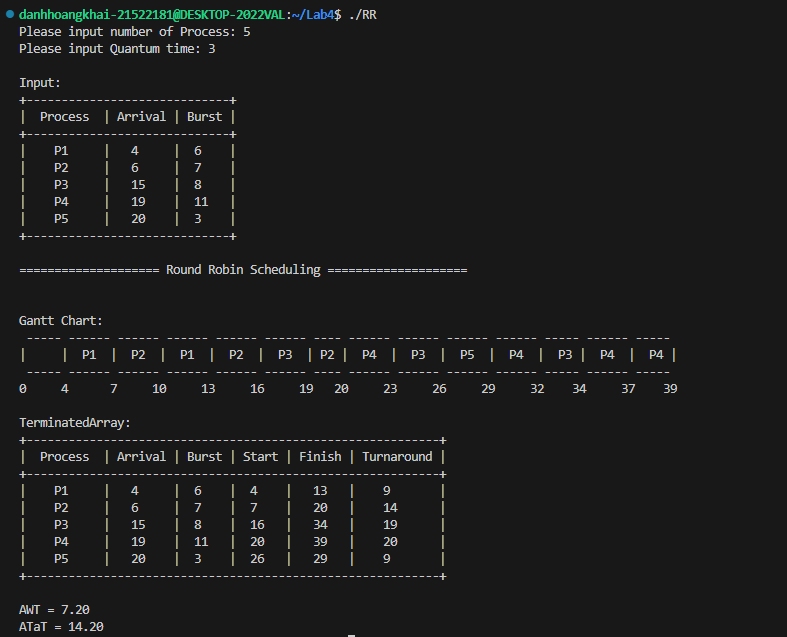
exportGanttChart(iTerminated, TerminatedArray); quickSort(PrintArray, 0, iPrint - 1, SORT\_BY\_PID); printf("\nTerminatedArray:\n"); printProcess(iPrint, PrintArray); calculateAWT(iPrint, PrintArray); calculateATaT(iPrint, PrintArray);

return 0;

}

# 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.

Test case 1



*Chạy tay test case*

Test case 2

A screenshot of a computer

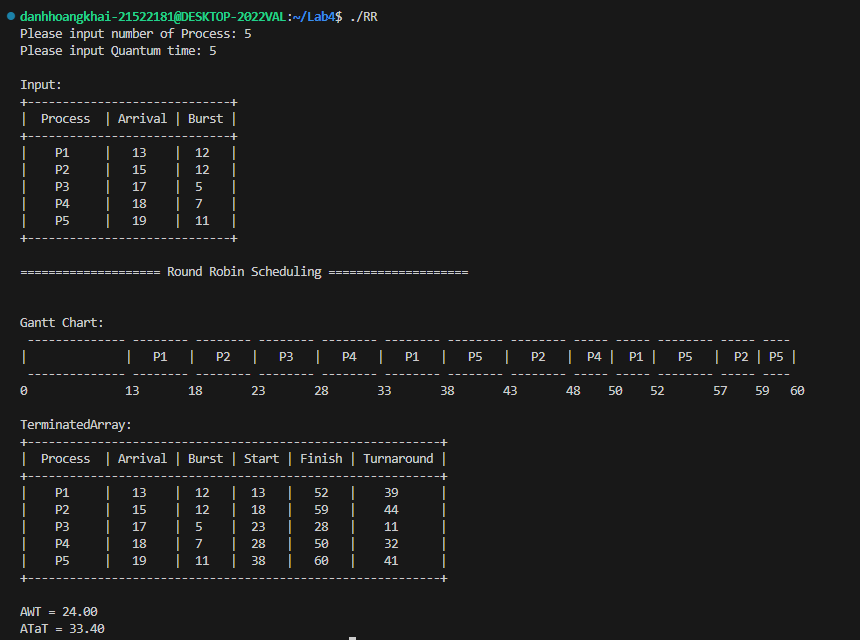
Description automatically generated

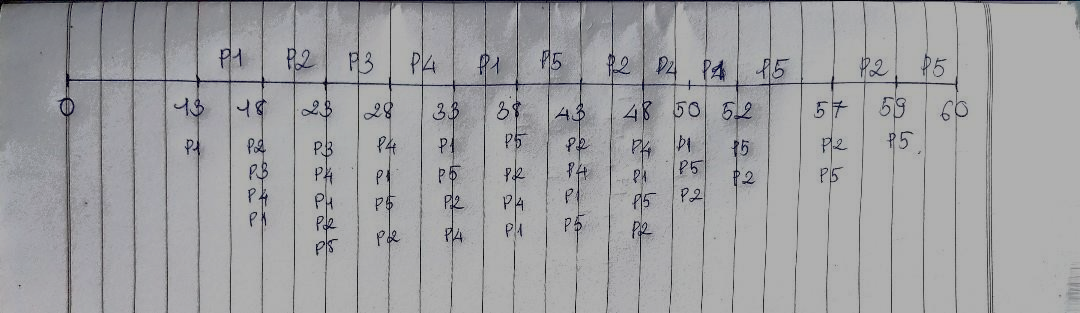
A close-up of a graph

Description automatically generated

*Chạy tay test case 2*

Test case 3





Chạy tay test case 3