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

HỆ ĐIỀU HÀNH  
BÁO CÁO LAB 4

**CHECKLIST**

**3.5. BÀI TẬP THỰC HÀNH**

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

**3.6. BÀI TẬP ÔN TẬP**

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

**Tự chấm điểm:** 10/10

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

**2.5. BÀI TẬP THỰC HÀNH**

# Giải thuật Shortest-Job-First

**Vẽ lưu đồ thuật toán:**

A diagram with a diagram and a diagram

Description automatically generated with medium confidence

**Chạy tay 1 test case:**  
*test case 1:*

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

A green rectangular with black numbers

Description automatically generated

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

**CODE:**

1. #include <stdio.h>
2. #include <stdlib.h>
4. **typedef** **struct** {
5. **int** iPID;
6. **int** iArrival, iBurst;
7. **int** iStart, iFinish, iWaiting, iResponse, iTaT;
8. **int** iRemaining;
9. } PCB;
11. **void** inputProcess(**int** n, PCB P[]) {
12. **for** (**int** i = 0; i < n; i++) {
13. printf("Input Process %d\n", i + 1);
14. printf("PID: ");
15. scanf("%d", &P[i].iPID);
16. printf("Arrival Time: ");
17. scanf("%d", &P[i].iArrival);
18. printf("Burst Time: ");
19. scanf("%d", &P[i].iBurst);
20. P[i].iRemaining = P[i].iBurst;
21. P[i].iResponse = -1;
22. }
23. }
25. **void** ExportGanttChart(**int** a[], **int** totalTime) {
26. printf("P%d\t", a[0]);
27. **for** (**int** iTime = 1; iTime <= totalTime; ++iTime) {
28. **if** (a[iTime] == a[iTime - 1]) **continue**;
29. printf("P%d\t", a[iTime]);
30. }
31. printf("\n0\t");
32. **for** (**int** iTime = 1; iTime <= totalTime; ++iTime) {
33. **if** (a[iTime] == a[iTime - 1]) **continue**;
34. printf("%d\t", iTime);
35. }
36. printf("%d\n", totalTime + 1);
37. }
39. **void** calculateAWT(**int** n, PCB P[]) {
40. **float** totalWaitingTime = 0;
41. **float** averageWaitingTime = 0;
43. **for** (**int** i = 0; i < n; i++) {
44. P[i].iWaiting = P[i].iStart - P[i].iArrival;
45. totalWaitingTime += P[i].iWaiting;
46. }
48. **if** (n > 0) {
49. averageWaitingTime = totalWaitingTime / n;
50. }
52. printf("Average Waiting Time: %.2f\n", averageWaitingTime);
53. }
55. **void** calculateATaT(**int** n, PCB P[]) {
56. **float** totalTurnaroundTime = 0;
57. **float** averageTurnaroundTime = 0;
59. **for** (**int** i = 0; i < n; i++) {
60. P[i].iTaT = P[i].iFinish - P[i].iArrival;
61. totalTurnaroundTime += P[i].iTaT;
62. }
64. **if** (n > 0) {
65. averageTurnaroundTime = totalTurnaroundTime / n;
66. }
68. printf("Average Turnaround Time: %.2f\n", averageTurnaroundTime);
69. }
71. **void** swapProcess(PCB \*P, PCB \*Q) {
72. PCB temp = \*P;
73. \*P = \*Q;
74. \*Q = temp;
75. }
77. **int** partition(PCB P[], **int** low, **int** high) {
78. **int** pivot = P[high].iBurst;
79. **int** i = low - 1;
81. **for** (**int** j = low; j <= high - 1; j++) {
82. **if** (P[j].iBurst < pivot) {
83. i++;
84. swapProcess(&P[i], &P[j]);
85. }
86. }
88. swapProcess(&P[i + 1], &P[high]);
89. **return** (i + 1);
90. }
92. **void** quickSort(PCB P[], **int** low, **int** high) {
93. **if** (low < high) {
94. **int** pi = partition(P, low, high);
96. quickSort(P, low, pi - 1);
97. quickSort(P, pi + 1, high);
98. }
99. }
101. **int** main() {
102. **int** iNumberOfProcess;
104. printf("Please input number of Processes: ");
105. scanf("%d", &iNumberOfProcess);
107. PCB Input[iNumberOfProcess];
109. inputProcess(iNumberOfProcess, Input);
110. quickSort(Input, 0, iNumberOfProcess - 1);
112. **int** currentTime = 0;
113. **int** remainingProcesses = iNumberOfProcess;
114. **int** selectedProcess = -1;
115. **int** minBurst = 999999; // Giá trị burst time nhỏ nhất hiện tại
117. **while** (remainingProcesses > 0) {
118. **for** (**int** i = 0; i < iNumberOfProcess; i++) {
119. **if** (Input[i].iArrival <= currentTime && Input[i].iRemaining > 0) {
120. **if** (Input[i].iBurst < minBurst) {
121. minBurst = Input[i].iBurst;
122. selectedProcess = i;
123. }
124. }
125. }
127. **if** (selectedProcess == -1) {
128. currentTime++;
129. } **else** {
130. Input[selectedProcess].iStart = currentTime;
131. Input[selectedProcess].iFinish =
132. Input[selectedProcess].iStart + Input[selectedProcess].iRemaining;
133. currentTime = Input[selectedProcess].iFinish;
134. Input[selectedProcess].iRemaining = 0;
135. remainingProcesses--;
136. minBurst = 999999;
137. selectedProcess = -1;
138. }
139. }
141. printf("\n===== SJF Scheduling =====\n");
142. calculateAWT(iNumberOfProcess, Input);
143. calculateATaT(iNumberOfProcess, Input);
145. // Tạo và hiển thị biểu đồ Gantt
146. **int** ganttChart[currentTime];
147. **for** (**int** i = 0; i < iNumberOfProcess; ++i) {
148. **for** (**int** j = Input[i].iStart; j < Input[i].iFinish; ++j) {
149. ganttChart[j] = Input[i].iPID;
150. }
151. }
153. ExportGanttChart(ganttChart, currentTime - 1);
155. **return** 0;
156. }

**Test case 1:**

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

A green rectangular with black numbers

Description automatically generated

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

A screenshot of a computer

Description automatically generated

**Test case 2:**

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

1. **SJF**

A graph with numbers and a line

Description automatically generated

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

A screenshot of a computer program

Description automatically generated

**Test case 3**

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

1. **SJF**

**A green rectangle with black text

Description automatically generated**

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

A screenshot of a computer program

Description automatically generated

# Giải thuật Shortest-Remaining-Time-First

Shortest-Remaining-Time-First

**Vẽ lưu đồ:**

**A diagram of a process

Description automatically generated**

**Chạy tay lưu đồ:**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define SORT\_BY\_ARRIVAL 0

#define SORT\_BY\_PID 1

#define SORT\_BY\_BURST 2

#define SORT\_BY\_START 3

#define SORT\_BY\_REMAINING 4

typedef struct

{

int iPID;

int iArrival, iBurst;

int iStart, iFinish, iWaiting, iResponse, iTaT;

int iRemaining, iLastRunning;

} PCB;

void inputProcess(int n, PCB P[])

{

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

printf("Input Process %d\n",i+1);

printf("PID: ");

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

printf("Arrival Time: ");

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

printf("Burst Time: ");

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

P[i].iRemaining = P[i].iBurst;

P[i].iResponse = -1;

}

}

void printProcess(int n, PCB P[])

{

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

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

printf("Arrival Time: %d\n",P[i].iArrival);

printf("Burst Time: %d\n",P[i].iBurst);

printf("Start Time: %d\n",P[i].iStart);

printf("Finish Time: %d\n",P[i].iFinish);

printf("Waiting Time: %d\n",P[i].iWaiting);

printf("Response Time: %d\n",P[i].iResponse);

printf("Turn Around Time: %d\n",P[i].iTaT);

}

}

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

{

P[\*n] = Q;

(\*n)++;

}

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

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

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

}

(\*n)--;

}

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

PCB temp = \*P;

\*P = \*Q;

\*Q = temp;

}

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

if (iCriteria == SORT\_BY\_ARRIVAL) {

int pivot = P[high].iArrival;

int i = (low-1);

for(int j = low; j <= high; j++)

{

if (P[j].iArrival < pivot)

{

i++;

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

}

}

swapProcess(&P[i+1], &P[high]);

return (i+1);

} else if (iCriteria == SORT\_BY\_PID) { // Sort by PID

int pivot = P[high].iPID;

int i = (low-1);

for(int j = low; j <= high; j++)

{

if (P[j].iPID < pivot)

{

i++;

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

}

}

swapProcess(&P[i+1], &P[high]);

return (i+1);

} else if (iCriteria == SORT\_BY\_BURST) {

int pivot = P[high].iBurst;

int i = (low-1);

for(int j = low; j <= high; j++)

{

if (P[j].iBurst < pivot)

{

i++;

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

}

}

swapProcess(&P[i+1], &P[high]);

return (i+1);

} else if (iCriteria == SORT\_BY\_START) {

int pivot = P[high].iStart;

int i = (low-1);

for(int j = low; j <= high; j++)

{

if (P[j].iStart < pivot)

{

i++;

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

}

}

swapProcess(&P[i+1], &P[high]);

return (i+1);

} else if (iCriteria == SORT\_BY\_REMAINING) {

int pivot = P[high].iRemaining;

int i = (low-1);

for(int j = low; j <= high; j++)

{

if (P[j].iRemaining < pivot)

{

i++;

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

}

}

swapProcess(&P[i+1], &P[high]);

return (i+1);

}

}

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

if(low < high) {

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

quickSort(P, low, pi-1, iCriteria);

quickSort(P, pi+1, high, iCriteria);

}

}

void calculateAWT(int n, PCB P[])

{

int sum = 0;

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

sum += P[i].iWaiting;

}

printf("Average Waiting Time: %f\n",(float)sum/n);

}

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

int sum = 0;

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

sum += P[i].iTaT;

}

printf("Average TAT Time: %f\n",(float)sum/n);

}

void ExportGanttChart(int a[], int totalTime) {

printf("P%d\t", a[0]);

for (int iTime = 1; iTime <= totalTime; ++iTime) {

if (a[iTime] == a[iTime-1]) continue;

printf("P%d\t", a[iTime]);

}

printf("\n0\t");

for (int iTime = 1; iTime <= totalTime; ++iTime) {

if (a[iTime] == a[iTime-1]) continue;

printf("%d\t", iTime);

}

printf("%d\n", totalTime+1);

}

int main()

{

PCB Input[10];

PCB ReadyQueue[10];

PCB TerminatedArray[10];

int iNumberOfProcess;

printf("Please input number of Process: ");

scanf("%d", &iNumberOfProcess);

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

inputProcess(iNumberOfProcess, Input);

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

// SRT Algorithm Implementation

// Push to ready queue

int totalTime = 0;

for (int i = 0; i < iNumberOfProcess; ++i) {

totalTime += Input[0].iBurst;

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

removeProcess(&iRemain, 0, Input);

}

// Executing SRT Algorithm

int ganttChart[totalTime];

for (int currentTime = 0; currentTime <= totalTime-1; ++currentTime) {

int i;

for (i = 0; i < iReady; ++i) {

if (ReadyQueue[i].iArrival <= currentTime) break;

}

// Pick the current process (which has a shortest remaining time)

PCB\* currentProcess = &ReadyQueue[i];

// Store execution information to Gantt Chart

ganttChart[currentTime] = currentProcess -> iPID;

if (currentProcess -> iResponse == -1) {

currentProcess -> iStart = currentTime;

currentProcess -> iResponse = currentProcess -> iStart - currentProcess -> iArrival;

currentProcess -> iWaiting = currentProcess -> iResponse;

currentProcess -> iLastRunning = currentTime;

}

else {

currentProcess -> iWaiting += currentTime - currentProcess -> iLastRunning - 1;

currentProcess -> iLastRunning = currentTime;

}

currentProcess -> iRemaining--;

if (currentProcess -> iRemaining == 0) {

currentProcess -> iFinish = currentTime+1;

currentProcess -> iTaT = currentTime+1 - currentProcess -> iArrival;

pushProcess(&iTerminated, TerminatedArray, \*currentProcess);

removeProcess(&iReady, 0, ReadyQueue);

}

// Sort the ready queue

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

}

// Output

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

ExportGanttChart(ganttChart, totalTime-1);

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

calculateAWT(iTerminated, TerminatedArray);

calculateATaT(iTerminated, TerminatedArray);

return 0;

}

**Test case 1: Chạy tay**

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

A green rectangular object with black text

Description automatically generated

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

**Chạy code:**

A screenshot of a computer program

Description automatically generated

**Test case 2: Chạy tay**

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

**A graph with a line

Description automatically generated with medium confidence**

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

**Chạy code:**

A screenshot of a computer

Description automatically generated

**Test case 3: Chạy tay**

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

A green rectangular object with black text

Description automatically generated

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

**Chạy code:**

**A screenshot of a computer

Description automatically generated**

**2.6. BÀI TẬP ÔN TẬP**

# Giải thuật Round Robin

Trả lời...

A diagram of a process

Description automatically generated

**Test case 1:**

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

**RR (q = 4)**

A green rectangular with black text

Description automatically generated

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

**Code:   
A screen shot of a computer program

Description automatically generated  
A screen shot of a computer

Description automatically generated** **A screen shot of a computer code

Description automatically generated** A computer screen with text and numbers

Description automatically generated A computer screen shot of a program code

Description automatically generated **A computer screen shot of code

Description automatically generated**A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

**A computer screen shot of a program

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

**Test case 1:**

**A screenshot of a computer program

Description automatically generated**

**A green and black text with black text

Description automatically generated with medium confidence**

**Test case 2:**

A graph of a number

Description automatically generatedA screen shot of a computer

Description automatically generated

Test case 3:

A screenshot of a computer

Description automatically generated  
A graph with numbers and a bar chart

Description automatically generated with medium confidence