4

Lab

**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:

A screenshot of a computer

Description automatically generated

**BÀI TẬP ÔN TẬP**

**A – GIẢI THUẬT SJF**

1. Source code:
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<time.h>
5. typedef struct {
6. int iArrival, iBurst, iPID;
7. int iResponse, iWaiting, iStart, iFinish, iTaT;
8. }PCB;
9. void INIT\_ARR(int Num\_Of\_Process, PCB P[]){
10. for(int i=0; i<Num\_Of\_Process; i++){
11. P[i].iPID = i + 1;
12. printf("Process %d: \n", i+1);
13. printf("Arrival: ");
14. scanf("%d", &P[i].iArrival);
15. printf("Burst: ");
16. scanf("%d", &P[i].iBurst);
17. }
18. }
19. void PRINT\_ARR(int Num\_Of\_Process, PCB P[]){
20. for(int i=0; i<Num\_Of\_Process; i++){
21. printf("P%d ", P[i].iPID);
22. printf("(%d, ", P[i].iArrival);
23. printf("%d)\n", P[i].iBurst);
24. }
25. }
26. void swap(PCB \*A, PCB \*B){
27. PCB Temp = \*A;
28. \*A = \*B;
29. \*B = Temp;
30. }
31. void quickSort\_Arrival(PCB a[], int l, int r){
32. PCB p=a[(l+r)/2]; ///p: la phan tu privot
33. int i=l, j=r;
34. while(i<=j){
35. while(a[i].iArrival < p.iArrival){
36. i++;
37. }
38. while(a[j].iArrival > p.iArrival){
39. j--;
40. }
41. if(i<=j){
42. swap(&a[i],&a[j]);
43. i++;
44. j--;
45. }
46. }
47. if(i<r){
48. quickSort\_Arrival(a,i,r);
49. }
50. if(l<j){
51. quickSort\_Arrival(a,l,j);
52. }
53. }
54. void quickSort\_Burst(PCB a[], int l, int r){
55. PCB p=a[(l+r)/2]; ///p: la phan tu privot
56. int i=l, j=r;
57. while(i<=j){
58. while(a[i].iBurst<p.iBurst){
59. i++;
60. }
61. while(a[j].iBurst>p.iBurst){
62. j--;
63. }
64. if(i<=j){
65. swap(&a[i],&a[j]);
66. i++;
67. j--;
68. }
69. }
70. if(i<r){
71. quickSort\_Burst(a,i,r);
72. }
73. if(l<j){
74. quickSort\_Burst(a,l,j);
75. }
76. }
77. void UPDATE\_START(int P\_Terminated, PCB Terminated\_Arr[], PCB \*Q, int \*Time\_line){
78. if(P\_Terminated == 0){
79. Q->iStart = Q->iArrival;
80. }
81. else if(P\_Terminated > 0){
82. if(Q->iArrival <= Terminated\_Arr[P\_Terminated-1].iFinish){
83. Q->iStart = Terminated\_Arr[P\_Terminated-1].iFinish;
84. }
85. else if(Q->iArrival > Terminated\_Arr[P\_Terminated-1].iFinish){
86. Q->iStart = Q->iArrival;
87. }
88. }
89. \*Time\_line = Q->iStart;
90. }
91. void PUSH\_PROCESS(int \*n, PCB P[], PCB Q){
92. P[\*n] = Q;
93. \*n = \*n + 1;
94. }
95. void REMOVE\_PROCESS(int \*n, int index, PCB P[]){
96. for(int i=index; i<\*n-1; i++){
97. swap(&P[i], &P[i+1]);
98. }
99. \*n = \*n - 1;
100. }
101. void PRINT\_TERMINATED(int P\_Terminated, PCB Terminated\_Arr[]){
102. for(int i=0; i<P\_Terminated; i++){
103. printf("P%d: ", Terminated\_Arr[i].iPID);
104. printf("Start: %d, ", Terminated\_Arr[i].iStart);
105. printf("Finish: %d\n", Terminated\_Arr[i].iFinish);
106. }
107. }
108. void CALCULATE\_VAL(int n, PCB P[]){
109. double AVR\_WAITING = 0;
110. double AVR\_EXE = 0;
111. double AVR\_RESPONSE = 0;
112. for(int i=0; i<n; i++){
113. AVR\_RESPONSE += (P[i].iStart - P[i].iArrival)\*1.0;
114. AVR\_WAITING += (P[i].iStart - P[i].iArrival)\*1.0;
115. AVR\_EXE += (P[i].iFinish - P[i].iArrival)\*1.0;
116. }
117. printf("AVR Response: %f\n", AVR\_RESPONSE/n);
118. printf("AVR Waiting: %f\n", AVR\_WAITING/n);
119. printf("AVR TaT: %f\n", AVR\_EXE/n);
120. }
121. void DRAW\_GANTT(int Num\_Of\_Fragment, PCB Gantt[]){
122. if(Gantt[0].iStart != 0){
123. printf("{0}=");
124. printf("{%d}=== ", Gantt[0].iStart);
125. printf("P%d ===", Gantt[0].iPID);
126. }
127. else{
128. printf("{%d}=== ", Gantt[0].iStart);
129. printf("P%d ===", Gantt[0].iPID);
130. }
131. for(int i=1; i<Num\_Of\_Fragment; i++){
132. if(i != Num\_Of\_Fragment-1){
133. if(Gantt[i].iStart == Gantt[i-1].iFinish){
134. printf("{%d}=== ", Gantt[i].iStart);
135. printf("P%d ===", Gantt[i].iPID);
136. }
137. else if (Gantt[i-1].iFinish < Gantt[i].iStart){
138. printf("{%d}=", Gantt[i-1].iFinish);
139. printf("{%d}=== ", Gantt[i].iStart);
140. printf("P%d ===", Gantt[i].iPID);
141. }
142. }
143. else if (i == Num\_Of\_Fragment-1){
144. if(Gantt[i].iStart == Gantt[i-1].iFinish){
145. printf("{%d}=== ", Gantt[i].iStart);
146. printf("P%d ===", Gantt[i].iPID);
147. printf("{%d}", Gantt[i].iFinish);
148. }
149. else if(Gantt[i-1].iFinish < Gantt[i].iStart){
150. printf("{%d}=", Gantt[i-1].iFinish);
151. printf("{%d}=== ", Gantt[i].iStart);
152. printf("P%d ===", Gantt[i].iPID);
153. printf("{%d}", Gantt[i].iFinish);
154. }
155. }
156. }
157. printf("\n");
158. }
159. void AUTO\_INIT\_PROCESS(PCB P[], int n){
160. srand(time(0));
161. for(int i = 0; i < n; i++){
162. P[i].iPID = i+1;
163. P[i].iArrival = rand() % 21;
164. P[i].iBurst = rand() % 11 + 2;
165. }
166. }
167. void SORT\_ARRIVAL(int n, PCB P[]){
168. for(int j = 0; j < n; j++){
169. for(int i = 0; i < n-1; i++){
170. if(P[i].iBurst == P[i+1].iBurst && P[i].iArrival > P[i+1].iArrival){
171. swap(&P[i], &P[i+1]);
172. }
173. else{
174. continue;
175. }
176. }
177. }
179. }
180. int main(){
181. PCB Input[10];
182. PCB ReadyQueue[10];
183. PCB Terminated\_Arr[10];
184. int Num\_Of\_Process, P\_Ready = 0, P\_Terminated = 0;
185. printf("Nhap so luong Process: ");
186. scanf("%d", &Num\_Of\_Process);
187. int P\_Remain = Num\_Of\_Process;
188. int SELECT = 0;
189. printf("Khởi tạo mảng Process tự động [0]\n");
190. printf("Khởi tạo thủ công [1]\n");
191. printf("Lựa chọn: \n");
192. scanf("%d", &SELECT);
193. if(SELECT == 1){
194. INIT\_ARR(Num\_Of\_Process, Input);
195. }
196. else{
197. AUTO\_INIT\_PROCESS(Input, Num\_Of\_Process);
198. }
199. //
200. printf("------------Input-----------\n");
201. PRINT\_ARR(Num\_Of\_Process, Input);
202. quickSort\_Arrival(Input, 0, Num\_Of\_Process-1);
203. //
204. int Time\_line = 0;
205. PUSH\_PROCESS(&P\_Ready, ReadyQueue, Input[0]);
206. REMOVE\_PROCESS(&P\_Remain, 0, Input);
207. //Cap nhat Time\_line luc nay se bang thoi gian Arrival cua Process dau tien nap vao
208. Time\_line = ReadyQueue[0].iArrival;
209. //Khi nao cac chuong trinh chua duoc xu li xong thi thuat toan van chay
210. while(P\_Terminated < Num\_Of\_Process){
211. //Neu Time\_line lon hon Arrival cua cac Process trong Input thi nap no vao ReadyQueue
212. while(Time\_line >= Input[0].iArrival && P\_Remain > 0){
213. PUSH\_PROCESS(&P\_Ready, ReadyQueue, Input[0]);
214. REMOVE\_PROCESS(&P\_Remain, 0, Input);
215. quickSort\_Burst(ReadyQueue, 0, P\_Ready - 1);
216. 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
217. }
218. //Kiem tra neu sau vong nay khong co Process nao trong ReadyQueue nhung Input con thi nap Process vao
219. if(P\_Ready == 0){
220. while(Time\_line <= Input[0].iArrival && P\_Remain > 0){
221. PUSH\_PROCESS(&P\_Ready, ReadyQueue, Input[0]);
222. REMOVE\_PROCESS(&P\_Remain, 0, Input);
223. Time\_line = ReadyQueue[P\_Remain - 1].iArrival;
224. quickSort\_Burst(ReadyQueue, 0, P\_Ready - 1);
225. 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
226. }
227. }
228. //Cap nhat iStart cho Process
229. UPDATE\_START(P\_Terminated, Terminated\_Arr, &ReadyQueue[0], &Time\_line);
230. //Sau vong nay chac chan phai co Process de xu li
231. if(P\_Ready > 0){
232. //Cap nhat iFinish
233. ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;
234. //Dua vao Terminated\_Arr
235. PUSH\_PROCESS(&P\_Terminated, Terminated\_Arr, ReadyQueue[0]);
236. //Remove khoi ReadyQueue
237. REMOVE\_PROCESS(&P\_Ready, 0, ReadyQueue);
238. //Cap nhat Time\_line
239. Time\_line = Terminated\_Arr[P\_Terminated - 1].iFinish;
240. }
241. }
242. printf("--------------TERMINATED-----------\n");
243. PRINT\_TERMINATED(P\_Terminated, Terminated\_Arr);
244. printf("==============GANTT==============\n");
245. DRAW\_GANTT(P\_Terminated, Terminated\_Arr);
246. printf("--------------AVR------------\n");
247. CALCULATE\_VAL(P\_Terminated, Terminated\_Arr);
248. return 0;
249. }

2. Thực thi:

Ví dụ:

A table with numbers and letters

Description automatically generated

A green rectangular object with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

**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:

A screenshot of a graph

Description automatically generated

Thực thi chương trình trên Linux:

A computer screen shot of a computer screen

Description automatically generated

**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:

A screenshot of a computer

Description automatically generated

Thực thi trên linux:

A screenshot of a computer

Description automatically generated