

# Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

19 March 2020

---

## UCS1411 - Operating Systems Laboratory

---

### Lab Exercise 7: Implementation of Banker's Algorithm (Deadlock Avoidance)

#### *Objective:*

Develop a C program to implement the banker's algorithm for deadlock avoidance.

#### *Code:*

Q.To write a C program to implement the banker's algorithm for deadlock avoidance.

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
```

```

6 struct Job{
7     char *PID;
8     int maxReq[100];
9     int alloc[100];
10    int need[100];
11 };
12
13 typedef struct Job Process;
14
15 void initialise(Process *p,int NoR){
16     p->PID=(char*)malloc(100);
17     for(int i =0;i<NoR;i++){
18         p->maxReq[i]=0;
19         p->alloc[i]=0;
20         p->need[i]=0;
21     }
22 }
23
24 void acceptProcess(Process *p,int NoR){
25
26     printf("\nEnter PID: ");scanf(" %s",p->PID);
27     printf("Enter Maximum vector for the process: ");
28     for(int i=0;i<NoR;i++)
29         scanf("%d",&p->maxReq[i]);
30
31     printf("Enter Allocation vector for the process: ");
32     for(int i=0;i<NoR;i++)
33         scanf("%d",&p->alloc[i]);
34
35     for(int i=0;i<NoR;i++)
36         p->need[i]=p->maxReq[i]-p->alloc[i];
37 }
38
39 int vectorCheck(int vec1[],int vec2[],int NoR){
40     for(int i=0;i<NoR;i++)
41         if(vec1[i]>vec2[i])
42             return 0;
43     return 1;
44 }
45
46 int checkFlag(int complete_flag[],int NoP){
47     for(int i=0;i<NoP;i++)
48         if(complete_flag[i]==0)
49             return 1;
50     return 0;

```

```

51 }
52
53
54 int safetySequence(Process p[],int NoP,int NoR,int *available
){
55     //Safe sequence
56     char *safeSeq[100];
57     for(int i=0;i<100;i++)
58         safeSeq[i]=(char*)malloc(100);
59
60     int ssctr=0,tmpssctr=0;
61
62     //List of complete processes
63     int complete_flag[100];
64
65     //Initialise all processes as incomplete
66     for(int i=0;i<NoP;i++)
67         complete_flag[i]=0;
68     for(int pno=0;pno<NoP;pno++){
69
70         if(pno==0)
71             tmpssctr=ssctr;
72         //Check if an incomplete process has its need less
than the available resources
73         if(vectorCheck(p[pno].need,available,NoR)&&!
complete_flag[pno]){
74
75             strcpy(safeSeq[ssctr++],p[pno].PID);
76
77             //Set process as complete
78             complete_flag[pno]=1;
79
80             //Add allocated resources to the available
resources
81             for(int i=0;i<NoR;i++)
82                 available[i]+=p[pno].alloc[i];
83         }
84         if(pno==NoP-1){
85             //Check if all processes are complete
86             if (checkFlag(complete_flag,NoP))
87                 pno=-1;
88             //If no incomplete process that has need less
than available can be found, break the loop
89             if(tmpssctr==ssctr)
90                 break;

```

```

91     }
92 }
93 if(tmpssctr==ssctr){
94     printf("\n The system is not in a safe state. \n");
95     return 0;
96 }
97 else{
98     printf("\n The safe sequence is: \n <");
99     for(int i=0;i<ssctr;i++){
100         printf(" %s ",safeSeq[i]);
101     }
102     printf(">\n");
103     printf("\n The system is in a safe state. \n");
104     return 1;
105 }
106 }
107
108 void RequestCheck(Process p[],int NoP,int NoR,char*
    reqProcess,int request[],int available[]){
109     int pno=0;
110     for(;pno<NoP;pno++){
111         if(strcmp(p[pno].PID,reqProcess)==0)
112             break;
113     }
114
115     //Check if the request made is less than the maximum
    resources that the process requires
116     if(vectorCheck(request,p[pno].need,NoR)){
117
118         //Check if the request made is less than the
    available resources
119         if(vectorCheck(request,available,NoR)){
120
121             //Pretend to satisfy the request
122             for(int i=0;i<NoR;i++){
123                 p[pno].alloc[i]+=request[i];
124                 p[pno].need[i]-=request[i];
125                 available[i]-=request[i];
126             }
127             //Check if the system is in a safe state
128             if(safetySequence(p,NoP,NoR,available))
129                 printf("\n %s's request can be granted
    immediately. \n",reqProcess);
130             else
131                 printf("\n %s's request cannot be granted

```

```

132         immediately. Process has to wait. \n",reqProcess);
133     }
134     else{
135         printf("\n Available resources not enough to
136         satisfy request. Process has to wait. \n");
137     }
138 }
139 else{
140     printf("\nRequest exceeding maximum specified need of
141     the process. \n");
142 }
143 }
144
145 int main(){
146     int NoP; //Number of processes
147     int NoR; //Number of resources
148
149     int maxInstance[100]; //Maximum instances of each
150     resource
151
152     int available[100]; //Available number of resources in
153     the system
154
155     Process p[100]; //Process list
156
157     int option;
158     do{
159         printf("\n
160         -----\n");
161         printf("\n
162                 BANKERS ALGORITHM
163                 \n");
164         printf("\n
165         -----\n");
166
167         printf("\n 1.Read data \n 2.Print data \n 3.Safety
168         Sequence \n 4.Request \n 0.Exit \n");
169         printf("\n Enter option: ");scanf("%d",&option);
170
171         if(option==1){
172             printf("\n Enter number of processes: ");scanf("%
173             d",&NoP);
174
175             printf("\n Enter number of resources: ");scanf("%
176             d",&NoR);

```

```

166
167         for(int i=0;i<NoR;i++){
168             printf("\n Enter maximum instance of resource
169             %c : ", 'A'+i);scanf("%d",&maxInstance[i]);
170         }
171
172         for(int i=0;i<NoP;i++){
173             initialise(&p[i],NoR);
174             acceptProcess(&p[i],NoR);
175         }
176
177         for(int i=0;i<NoR;i++){
178             int sum=0;
179             for(int j=0;j<NoP;j++){
180                 sum+=p[j].alloc[i];
181                 available[i]=maxInstance[i]-sum;
182             }
183         }
184         else if(option==2){
185             printf("%13s %13s %13s %13s\n","Max","Alloc","
186             Need","Available");
187             for(int i=0;i<4;i++){
188                 printf("%5s"," ");
189                 for(int j=0;j<NoR;j++){
190                     printf("%2c",'A'+j);
191                 }
192                 for(int j=0;j<=8-NoR*2;j++){
193                     printf(" ");
194                 }
195                 printf("\n");
196                 for(int i=0;i<NoP;i++){
197                     printf("%3s ",p[i].PID);
198                     for(int j=0;j<NoR;j++){
199                         printf("%2d",p[i].maxReq[j]);
200                     }
201                     for(int j=0;j<=8-NoR*2;j++){
202                         printf(" ");
203                     }
204                     printf("%5s"," ");
205                     for(int j=0;j<NoR;j++){
206                         printf("%2d",p[i].alloc[j]);
207                     }
208                     for(int j=0;j<=8-NoR*2;j++){
209                         printf(" ");

```

```

209         printf("%5s", " ");
210         for(int j=0; j<NoR; j++){
211             printf("%2d", p[i].need[j]);
212         }
213         for(int j=0; j<=8-NoR*2; j++)
214             printf(" ");
215
216         if(i==0){
217             printf("%5s", " ");
218             for(int j=0; j<NoR; j++){
219                 printf("%2d", available[j]);
220             }
221             for(int j=0; j<=8-NoR*2; j++)
222                 printf(" ");
223         }
224         printf("\n");
225     }
226 }
227 else if(option==3){
228     safetySequence(p, NoP, NoR, available);
229 }
230 else if(option==4){
231     for(int i=0; i<NoR; i++){
232         int sum=0;
233         for(int j=0; j<NoP; j++)
234             sum+=p[j].alloc[i];
235         available[i]=maxInstance[i]-sum;
236     }
237
238     char* reqProcess=(char*)malloc(100);
239
240     int request[100];
241
242     printf("\nEnter PID of process making request: ")
;scanf(" %s", reqProcess);
243     printf("\nEnter request vector: ");
244     for(int i=0; i<NoR; i++)
245         scanf("%d", &request[i]);
246
247     RequestCheck(p, NoP, NoR, reqProcess, request,
available);
248 }
249 else if(option!=0){
250     printf("\n Invalid input \n");
251 }

```

```

252         else;
253     }while(option);
254 }

```

### ***Output:***

```

1
2  -----
3
4                BANKERS  ALGORITHM
5  -----
6
7
8
9  1.Read data
10 2.Print data
11 3.Safety Sequence
12 4.Request
13 0.Exit
14
15 Enter option: 1
16
17 Enter number of processes: 5
18
19 Enter number of resources: 4
20
21 Enter maximum instance of resource A : 3
22
23 Enter maximum instance of resource B : 14
24
25 Enter maximum instance of resource C : 12
26
27 Enter maximum instance of resource D : 12
28
29 Enter PID: P0
30 Enter Maximum vector for the process: 0 0 1 2
31 Enter Allocation vector for the process: 0 0 1 2
32
33 Enter PID: P1
34 Enter Maximum vector for the process: 1 7 5 0
35 Enter Allocation vector for the process: 1 0 0 0
36
37 Enter PID: P2
38 Enter Maximum vector for the process: 2 3 5 6
39 Enter Allocation vector for the process: 1 3 5 4

```



```

40
41 Enter PID: P3
42 Enter Maximum vector for the process: 0 6 5 2
43 Enter Allocation vector for the process: 0 6 3 2
44
45 Enter PID: P4
46 Enter Maximum vector for the process: 0 6 5 6
47 Enter Allocation vector for the process: 0 0 1 4
48
49 -----
50
51 BANKERS ALGORITHM
52
53 -----
54
55
56 1.Read data
57 2.Print data
58 3.Safety Sequence
59 4.Request
60 0.Exit
61
62 Enter option: 2
63
64      Max      Alloc      Need      Available
65      A B C D      A B C D      A B C D      A B C D
66 P0      0 0 1 2      0 0 1 2      0 0 0 0      1 5 2 0
67 P1      1 7 5 0      1 0 0 0      0 7 5 0
68 P2      2 3 5 6      1 3 5 4      1 0 0 2
69 P3      0 6 5 2      0 6 3 2      0 0 2 0
70 P4      0 6 5 6      0 0 1 4      0 6 4 2
71
72 -----
73
74 BANKERS ALGORITHM
75
76 -----
77
78 1.Read data
79 2.Print data
80 3.Safety Sequence
81 4.Request
82 0.Exit
83
84 Enter option: 3

```

```

85
86 The safe sequence is:
87 < P0 P2 P3 P4 P1 >
88
89 The system is in a safe state.
90
91 -----
92
93 BANKERS ALGORITHM
94
95 -----
96
97
98 1.Read data
99 2.Print data
100 3.Safety Sequence
101 4.Request
102 0.Exit
103
104 Enter option: 4
105
106 Enter PID of process making request: P1
107
108 Enter request vector: 0 4 2 0
109
110 The safe sequence is:
111 < P0 P2 P3 P4 P1 >
112
113 The system is in a safe state.
114
115 P1's request can be granted immediately.
116
117 -----
118
119 BANKERS ALGORITHM
120
121 -----
122
123
124 1.Read data
125 2.Print data
126 3.Safety Sequence
127 4.Request
128 0.Exit
129

```

130 Enter option: 0

---