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.

```
#include < stdio.h>
#include < stdlib.h>
#include < string.h>
```

```
6 struct Job{
      char *PID;
      int maxReq[100];
      int alloc[100];
      int need[100];
10
11 };
13 typedef struct Job Process;
15 void initialise(Process *p,int NoR){
      p->PID=(char*)malloc(100);
      for(int i =0;i<NoR;i++){</pre>
17
           p->maxReq[i]=0;
           p->alloc[i]=0;
19
           p->need[i]=0;
21
      }
22 }
24 void acceptProcess(Process *p,int NoR){
25
      printf("\nEnter PID: ");scanf(" %s",p->PID);
      printf("Enter Maximum vector for the process: ");
27
      for(int i=0;i<NoR;i++)</pre>
           scanf("%d",&p->maxReq[i]);
29
      printf("Enter Allocation vector for the process: ");
31
      for(int i=0;i<NoR;i++)</pre>
           scanf("%d",&p->alloc[i]);
33
      for(int i=0;i<NoR;i++)</pre>
           p->need[i]=p->maxReq[i]-p->alloc[i];
36
37 }
38
39 int vectorCheck(int vec1[],int vec2[],int NoR){
      for(int i=0;i<NoR;i++)</pre>
40
           if (vec1[i]>vec2[i])
41
42
               return 0;
      return 1;
44 }
46 int checkFlag(int complete_flag[],int NoP){
      for(int i=0;i<NoP;i++)</pre>
           if (complete_flag[i] == 0)
48
               return 1;
      return 0;
50
```

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

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

```
immediately. Process has to wait. \n", reqProcess);
132
         else{
133
              printf("\n Available resources not enough to
     satisfy request. Process has to wait. \n");
      }
136
      else{
137
          printf("\nRequest exceeding maximum specified need of
138
      the process. \n");
139
140 }
141
142 int main(){
      int NoP; //Number of processes
      int NoR; //Number of resources
144
145
      int maxInstance[100]; //Maximum instances of each
146
     resource
147
     int available[100]; //Available number of resources in
     the system
      Process p[100];  //Process list
150
      int option;
      do{
154
         printf("\n
      ----\n"):
          printf("\n
                                   BANKERS ALGORITHM
156
             \n");
          printf("\n
157
      ----\n");
158
          printf("\n 1.Read data \n 2.Print data \n 3.Safety
     Sequence \n 4. Request \n 0. Exit \n");
          printf("\n Enter option: ");scanf("%d",&option);
161
          if (option == 1) {
              printf("\n Enter number of processes: ");scanf("%
163
     d", & NoP);
164
              printf("\n Enter number of resources: ");scanf("%
     d",&NoR);
```

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

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

```
}while(option);
253
254 }
  Output:
   _____
2
                   BANKERS ALGORITHM
9 1.Read data
10 2.Print data
11 3. Safety Sequence
   4.Request
12
13
   0.Exit
14
15 Enter option: 1
17 Enter number of processes: 5
19 Enter number of resources: 4
21 Enter maximum instance of resource A: 3
23 Enter maximum instance of resource B: 14
_{25} Enter maximum instance of resource C : 12
27 Enter maximum instance of resource D : 12
29 Enter PID: PO
_{30} Enter Maximum vector for the process: 0 0 1 2
31 Enter Allocation vector for the process: 0 0 1 2
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
37 Enter PID: P2
_{\rm 38} Enter Maximum vector for the process: 2 3 5 6
_{39} Enter Allocation vector for the process: 1 3 5 4
```

else:

```
40
41 Enter PID: P3
_{\rm 42} Enter Maximum vector for the process: 0 6 5 2
43 Enter Allocation vector for the process: 0 6 3 2
45 Enter PID: P4
_{\rm 46} Enter Maximum vector for the process: 0 6 5 6
47 Enter Allocation vector for the process: 0 0 1 4
50
                  BANKERS ALGORITHM
51
   _____
53
54
55
56 1.Read data
  2.Print data
58 3.Safety Sequence
  4.Request
  0.Exit
61
  Enter option: 2
         Max
                     Alloc
                                    Need
                                            Available
63
                                A B C D 0 0 0 0
       A B C D
                   A B C D
                                              ABCD
      0 0 1 2
65 PO
                    0 0 1 2
                                               1 5 2 0
     1 7 5 0
                   1 0 0 0
                                 0 7 5 0
66 P1
67 P2 2 3 5 6
                   1 3 5 4
                                 1 0 0 2
                    0 6 3 2
      0 6 5 2
                                  0 0 2 0
  P3
  P4
     0 6 5 6
                    0 0 1 4
                                  0 6 4 2
70
72
                  BANKERS ALGORITHM
73
74
76
78 1.Read data
  2.Print data
  3. Safety Sequence
  4.Request
82 O.Exit
84 Enter option: 3
```

```
85
   The safe sequence is:
   < P0 P2 P3 P4 P1 >
   The system is in a safe state.
89
91
92
                    BANKERS ALGORITHM
93
   1.Read data
   2.Print data
3. Safety Sequence
   4.Request
   0.Exit
102
   Enter option: 4
104
106 Enter PID of process making request: P1
108 Enter request vector: 0 4 2 0
   The safe sequence is:
110
   < P0 P2 P3 P4 P1 >
   The system is in a safe state.
113
114
   P1's request can be granted immediately.
115
117
118
                    BANKERS ALGORITHM
119
121
123
   1.Read data
   2.Print data
   3. Safety Sequence
127
   4.Request
   0.Exit
128
129
```

130 Enter option: 0