Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

25 March 2020

UCS1411 - Operating Systems Laboratory

Lab Exercise 8: Implementation of Memory Management Algorithms

Objective:

Develop a C program to implement the Memory Management Algorithms.

Code:

Q.To write a C program to implement the Memory Management Algorithms.

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

#define hole "H"

struct Box{
    int start_byte;
    int end_byte;
    int size;
    char* status;
    struct Box* next;
};

typedef struct Box Node;

void initialise(Node* n){
```

```
18
19      n->start_byte=-1;
20      n->end_byte=-1;
21      n->size=0;
22      n->status=(char*)malloc(10);
23      strcpy(n->status,hole);
24      n->next=NULL;
25  }
26
27  //Accept details of partitions
28  void readNode(Node *n){
29      printf("Enter the start and end address of partition: ");
30      scanf("%d",&n->start_byte);scanf("%d",&n->end_byte);
31
32      n->size=n->end_byte-n->start_byte;
33  }
34
35  //Copy one partition to another
36  void copyNode(Node *OG,Node *copy){
37      copy->start_byte=OG->start_byte;
38      copy->end_byte=OG->end_byte;
39      copy->size=OG->size;
40      strcpy(copy->status,OG->status);
41      copy->next=NULL;
42  }
43
44  //Count number of partitions
45  int partitionCount(Node* LL){
46      Node *tmp=LL;
47      int ctr=0;
48      tmp=tmp->next;
49      while(tmp){
50          ctr++;
51          tmp=tmp->next;
52      }
53      return ctr;
54  }
55
56  //Insert into Linked List
57  void insert(Node *LL, Node *n){
58
59      Node *stmp=LL;
60      while(stmp->next && n->start_byte >= stmp->next->start_byte){
61          stmp=stmp->next;
62      }
63      n->next=stmp->next;
64      stmp->next=n;
65  }
66
67  //Delete Node from Linked List
68  void delete(Node *LL,char *status){
69      Node *tmp=LL;
70      Node *prev=LL;
71      tmp=tmp->next;
72      while(strcmp(status,tmp->status)!=0){
73          prev=tmp;
74          tmp=tmp->next;
```

```
    }
    prev->next=tmp->next;
    free(tmp);
}

//Display memory
void displayMemory(Node *LL){

    int no_of_partitions=partitionCount(LL);

    Node *tmp=LL;
    tmp=tmp->next;

    if(tmp==NULL){
        printf("\nNULL\n");
    }
    else{
        //Display top line
        for(int i=0;i<no_of_partitions;i++){
            printf(" ---------------  ");
        }

        //Display order of processes
        printf("\n");
        while(tmp){
            printf("|_____%4s_____| ",tmp->status);
            tmp=tmp->next;
        }
        printf("\n");

        //Display time line
        tmp=LL;
        tmp=tmp->next;
        while(tmp){
            printf("%-8d%8d ",tmp->start_byte,tmp->end_byte);
            tmp=tmp->next;
        }
        printf("\n\n");
    }
}
```

```

#include "LinkedList.h"

//Select first hole that is able to satisfy the process
void FirstFitInsert(Node *AllocLL, Node *FreePoolLL,Node *newNode){

    Node *tmp=FreePoolLL;
    Node *prev=tmp;
    tmp=tmp->next;

    //To determine whether a process can be allotted memory or not
    int entry=0;
    while(tmp){

        if(tmp->size < newNode->size){
            prev=tmp;
            tmp=tmp->next;
```

```
18          }
19          else{
20              newNode->start_byte=tmp->start_byte;
21              newNode->end_byte=newNode->start_byte + newNode->size;
22
23              //Check if size required is exactly the same size as a
    partition
24              if(newNode->size==tmp->size){
25                  prev->next=tmp->next;
26                  free(tmp);
27              }
28              else{
29                  tmp->start_byte = newNode->end_byte;
30                  tmp->size -= newNode->size;
31              }
32
33              insert(AllocLL,newNode);
34              entry=1;
35              break;
36          }
37      }
38      if(entry==0){
39          printf("\n Process %s cannot be allocated memory. \n",
    newNode->status);
40      }
41  }
42
43  //Select the smallest hole that can satisfy the process
44  void BestFitInsert(Node *AllocLL,Node *FreePoolLL,Node *newNode){
45
46      Node *tmp=FreePoolLL;
47      Node *prev=tmp;
48
49      //Consider the smallest hole out of all possible ones
50      Node *maybe=(Node*)malloc(sizeof(Node));
51      initialise(maybe);
52
53      tmp=tmp->next;
54
55      //To determine whether a process can be allotted memory or not
56      int entry=0;
57
58      //Determine possible value of maybe, and hence check if process
     can be allotted memory
59      while(tmp){
60          if(tmp->size < newNode->size){
61              tmp=tmp->next;
62          }
63          else{
64              maybe=tmp;
65              entry=1;
66              break;
67          }
68      }
69
70      if(entry){
71          tmp=FreePoolLL;
```

```
72          tmp=tmp->next;
73
74          //Find smallest hole
75          while(tmp){
76              if(tmp->size < newNode->size){
77                  prev=tmp;
78              }
79              else{
80                  if(tmp->size < maybe->size)
81                      maybe=tmp;
82              }
83              tmp=tmp->next;
84          }
85
86          newNode->start_byte=maybe->start_byte;
87          newNode->end_byte=newNode->start_byte + newNode->size;
88
89          //Check if size required is exactly the same as that of a
        partition
90          if(newNode->size==maybe->size){
91              prev->next=maybe->next;
92              free(maybe);
93          }
94          else{
95              maybe->start_byte = newNode->end_byte;
96              maybe->size -= newNode->size;
97          }
98
99          insert(AllocLL,newNode);
100     }
101     else{
102         printf("\n Process %s cannot be allocated memory. \n",
        newNode->status);
103     }
104 }
105
106 //Select largest hole that can satisfy the process
107 void WorstFitInsert(Node *AllocLL,Node *FreePoolLL,Node *newNode){
108
109     Node *tmp=FreePoolLL;
110     Node *prev=tmp;
111
112     //Consider the largest hole out of all possible ones
113     Node *maybe=(Node*)malloc(sizeof(Node));
114     initialise(maybe);
115
116     tmp=tmp->next;
117
118     //To determine whether a process can be allotted memory or not
119     int entry=0;
120
121     //Determine possible value of maybe, and hence check if process
         can be allotted memory
122     while(tmp){
123         if(tmp->size < newNode->size){
124             tmp=tmp->next;
125         }
```

```
126         else{
127             maybe=tmp;
128             entry=1;
129             break;
130         }
131     }
132 
133     if(entry){
134 
135         tmp=FreePoolLL;
136         tmp=tmp->next;
137 
138         //Find largest hole
139         while(tmp){
140             if(tmp->size < newNode->size){
141                 prev=tmp;
142             }
143             else{
144                 if(tmp->size > maybe->size)
145                     maybe=tmp;
146             }
147             tmp=tmp->next;
148         }
149 
150         newNode->start_byte=maybe->start_byte;
151         newNode->end_byte=newNode->start_byte + newNode->size;
152 
153         //Check if size required is exactly the same as that of a
    partition
154         if(newNode->size==maybe->size){
155             prev->next=maybe->next;
156             free(maybe);
157         }
158         else{
159             maybe->start_byte = newNode->end_byte;
160             maybe->size -= newNode->size;
161         }
162 
163         insert(AllocLL,newNode);
164     }
165     else{
166         printf("\n Process %s cannot be allocated memory. \n",
    newNode->status);
167     }
168 }
169 
170 void DeallocMem(Node *AllocLL,Node *FreePoolLL,char *status){
171 
172     Node *tmp=AllocLL;
173     Node *prev=tmp;
174     tmp=tmp->next;
175 
176     while(tmp&&strcmp(status,tmp->status)!=0){
177 
178         prev=tmp;
179         tmp=tmp->next;
180     }
```

```
181
182     //Copy contents of Node to be deleted to another node
183     prev->next=tmp->next;
184     Node *ftmp=(Node*)malloc(sizeof(Node));
185     initialise(ftmp);
186
187     copyNode(tmp,ftmp);
188
189     free(tmp);
190     strcpy(ftmp->status,hole);
191     insert(FreePoolLL,ftmp);
192 }
193
194
195 void createPhysicalMem(Node *AllocLL,Node *FreePoolLL){
196
197     //Physical Memory
198     Node *PM=(Node*)malloc(sizeof(Node));
199     initialise(PM);
200
201     Node *fp=FreePoolLL;Node *all=AllocLL;
202     fp=fp->next;all=all->next;
203
204     //Perform merge sort
205     while(fp && all){
206         Node *ftmp=(Node*)malloc(sizeof(Node));
207         initialise(ftmp);
208         copyNode(fp,ftmp);
209         Node *atmp=(Node*)malloc(sizeof(Node));
210         initialise(atmp);
211         copyNode(all,atmp);
212
213         if(fp->start_byte <= all->start_byte){
214
215             insert(PM,ftmp);
216             fp=fp->next;
217         }
218         else{
219
220             insert(PM,atmp);
221             all=all->next;
222         }
223     }
224
225     while(fp){
226         Node *ftmp=(Node*)malloc(sizeof(Node));
227         initialise(ftmp);
228         copyNode(fp,ftmp);
229
230         insert(PM,ftmp);
231         fp=fp->next;
232     }
233     while(all){
234         Node *atmp=(Node*)malloc(sizeof(Node));
235         initialise(atmp);
236         copyNode(all,atmp);
237
```

```
238         insert(PM,atmp);
239         all=all->next;
240     }
241     displayMemory(PM);
242 
243 }
244 
245 void holeJoin(Node *FreePoolLL){
246 
247     Node *tmp=FreePoolLL;
248     Node *prev=tmp;
249     tmp=tmp->next;
250 
251     while(tmp){
252         if(tmp->start_byte == prev->end_byte){
253             prev->end_byte = tmp->end_byte;
254             prev->size += tmp->size;
255             prev->next=tmp->next;
256             free(tmp);
257             tmp=prev;
258         }
259         prev=tmp;
260         tmp=tmp->next;
261 
262     }
263 }
264 
265 void main(){
266 
267     int no_of_partitions;
268 
269     Node *AllocLL=(Node*)malloc(sizeof(Node));
270     initialise(AllocLL);
271 
272     Node *FreePoolLL=(Node*)malloc(sizeof(Node));
273     initialise(FreePoolLL);
274 
275     printf("\n Enter Memory representation:\n");
276     printf("\n Enter number of partitons: ");scanf("%d",&
    no_of_partitions);
277 
278     for(int i=0;i<no_of_partitions;i++){
279         Node* newNode=(Node*)malloc(sizeof(Node));
280         initialise(newNode);
281         readNode(newNode);
282         insert(FreePoolLL,newNode);
283     }
284     int FitOpt;
285     int operation;
286     do{
287 
288         printf("\n Choose the memory allocation algorithm: \n");
289         printf(" 1.First Fit \n 2.Worst Fit \n 3.Best Fit \n 0.
    Exit \n");
290         printf(" Your choice: ");scanf("%d",&FitOpt);
291         if(FitOpt){
292             do{
```

```
293
294                 printf("\n Choose operation: \n 1.Process Entry \n
    2.Process Exit \n 3.Display \n");
295                 printf(" 4.Coalescing holes \n 0.Back \n Your
    choice: ");
296                 scanf("%d",&operation);
297
298                 if(operation==1){
299                     Node *process=(Node*)malloc(sizeof(Node));
300                     initialise(process);
301                     printf("\n Enter process ID: ");scanf(" %s",
    process->status);
302                     printf("\n Enter size required: ");scanf("%d",&
    process->size);
303                     if(FitOpt==1){
304                         FirstFitInsert(AllocLL,FreePoolLL,process);
305                     }
306                     else if(FitOpt==2){
307                         WorstFitInsert(AllocLL,FreePoolLL,process);
308                     }
309                     else if(FitOpt==3){
310                         BestFitInsert(AllocLL,FreePoolLL,process);
311                     }
312                     else if(FitOpt!=0){
313                         printf("\n Invalid algorithm. \n");
314                     }
315                     else;
316                 }
317                 else if(operation==2){
318                     char *status=(char*)malloc(100);
319                     printf("\n Enter process ID: ");scanf(" %s",
    status);
320                     DeallocMem(AllocLL,FreePoolLL,status);
321                 }
322                 else if(operation==3){
323                     printf("\n Allocated Memory: \n");
324                     displayMemory(AllocLL);
325                     printf("\n Free Pool: \n");
326                     displayMemory(FreePoolLL);
327                     printf("\n Physical Memory: \n");
328                     createPhysicalMem(AllocLL,FreePoolLL);
329                 }
330                 else if(operation==4){
331                     holeJoin(FreePoolLL);
332                     printf("\n Free Pool: \n");
333                     displayMemory(FreePoolLL);
334                 }
335                 else if(operation!=0){
336                     printf("\n Invalid operation. \n");
337                 }
338                 else;
339
340             }while(operation);
341         }
342     }while(FitOpt);
343 }
```

Output:

```
1
2  Enter Memory representation:
3
4  Enter number of partitons: 5
5 Enter the start and end address of partition: 0 100
6 Enter the start and end address of partition: 100 600
7 Enter the start and end address of partition: 600 800
8 Enter the start and end address of partition: 800 1100
9 Enter the start and end address of partition: 1100 1700
10
11  Choose the memory allocation algorithm:
12  1.First Fit
13  2.Worst Fit
14  3.Best Fit
15  0. Exit
16  Your choice: 1
17
18  Choose operation:
19  1.Process Entry
20  2.Process Exit
21  3.Display
22  4.Coalescing holes
23  0.Back
24  Your choice: 1
25
26  Enter process ID: P1
27
28  Enter size required: 212
29
30  Choose operation:
31  1.Process Entry
32  2.Process Exit
33  3.Display
34  4.Coalescing holes
35  0.Back
36  Your choice: 1
37
38  Enter process ID: P2
39
40  Enter size required: 417
41
42  Choose operation:
43  1.Process Entry
44  2.Process Exit
45  3.Display
46  4.Coalescing holes
47  0.Back
48  Your choice: 1
49
50  Enter process ID: p3
51
52  Enter size required: 112
53
54  Choose operation:
55  1.Process Entry
56  2.Process Exit
```

```
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P4

 Enter size required: 426

 Process P4 cannot be allocated memory.

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:
 --------------   --------------   --------------
|_____  P1_____| |_____  p3_____| |_____  P2_____|
100          312 312          424 1100        1517


 Free Pool:
 --------------   --------------   --------------   --------------
|_____    H_____| |_____    H_____| |_____    H_____| |_____    H_____|
0            100 424          600 600          800 800         1100
 --------------
|_____    H_____|
1517        1700
 Physical Memory:
 --------------   --------------   --------------   --------------
|_____    H_____| |_____  P1_____| |_____  p3_____| |_____    H_____|
0            100 100          312 312          424 424          600
 --------------   --------------   --------------   --------------
|_____    H_____| |_____    H_____| |_____  P2_____| |_____    H_____|
600          800 800         1100 1100        1517 1517        1700
 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

```

```
 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: p3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:

NULL

 Free Pool:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          312 312          424 424          600
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
600          800 800         1100 1100        1517 1517        1700
 Physical Memory:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          312 312          424 424          600
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
600          800 800         1100 1100        1517 1517        1700
 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0

 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
```

```
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P1

Enter size required: 212


Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P2

Enter size required: 417

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P3

Enter size required: 112

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P4

Enter size required: 426

Process P4 cannot be allocated memory.

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 3

Allocated Memory:
--------------   --------------   --------------
```

```
|_____  P2_____| |_____  P1_____| |_____  P3_____|
100           517 1100         1312 1312         1424


 Free Pool:
 ______________   ______________   ______________   ______________
|_____    H_____| |_____    H_____| |_____    H_____| |_____    H_____|
0             100 517           600 600           800 800          1100
 ______________
|_____    H_____|
1424         1700

 Physical Memory:
 ______________   ______________   ______________   ______________
|_____    H_____| |_____  P2_____| |_____    H_____| |_____    H_____|
0             100 100           517 517           600 600           800
 ______________   ______________   ______________   ______________
|_____    H_____| |_____  P1_____| |_____  P3_____| |_____    H_____|
800           1100 1100         1312 1312         1424 1424         1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
```

```
 Your choice: 3

 Allocated Memory:

NULL

 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0             100 100           517 517           600 600           800
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
800           1100 1100         1312 1312         1424 1424         1700

 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0             100 100           517 517           600 600           800
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
800           1100 1100         1312 1312         1424 1424         1700

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0


 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P1

 Enter size required: 212

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1
```

```
 Enter process ID: P2

 Enter size required: 417

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P3

 Enter size required: 112

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P4

 Enter size required: 426

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:
 --------------   --------------   --------------   --------------
|_____  P2_____| |_____  P3_____| |_____  P1_____| |_____  P4_____|
100           517 600           712 800          1012 1100         1526


 Free Pool:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0             100 517           600 712           800 1012         1100
 --------------
|_____   H_____|
1526        1700

Physical Memory:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____  P2_____| |_____   H_____| |_____  P3_____|
0             100 100           517 517           600 600           712
 --------------   --------------   --------------   --------------
      --------------
```

```
|_____   H_____| |_____  P1_____| |_____   H_____| |_____  P4_____|
      |_____   H_____|
712           800 800          1012 1012         1100 1100         1526
     1526         1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P4

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:

NULL
```

```
 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0             100 100           517 517           600 600           712
 ______________   ______________   ______________   ______________
       ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
      |_____   H_____|
712           800 800           1012 1012         1100 1100         1526
     1526         1700


 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0             100 100           517 517           600 600           712
 ______________   ______________   ______________   ______________
       ______________
|_____   H_____| |_____  P1_____| |_____   H_____| |_____  P4_____|
      |_____   H_____|
712           800 800           1012 1012         1100 1100         1526
     1526         1700

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 4

 Free Pool:
 ______________
|_____   H_____|
0            1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0

 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 0
```