

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.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define hole "H"
6
7 struct Box{
8     int start_byte;
9     int end_byte;
10    int size;
11    char* status;
12    struct Box* next;
13 };
14
15 typedef struct Box Node;
16
17 void initialise(Node* n){
```

```

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

```

```

75     }
76     prev->next=tmp->next;
77     free(tmp);
78 }
79
80 //Display memory
81 void displayMemory(Node *LL){
82
83     int no_of_partitions=partitionCount(LL);
84
85     Node *tmp=LL;
86     tmp=tmp->next;
87
88     if(tmp==NULL){
89         printf("\nNULL\n");
90     }
91     else{
92         //Display top line
93         for(int i=0;i<no_of_partitions;i++){
94             printf(" ----- ");
95         }
96
97         //Display order of processes
98         printf("\n");
99         while(tmp){
100             printf("|_____%4s____| ",tmp->status);
101             tmp=tmp->next;
102         }
103         printf("\n");
104
105         //Display time line
106         tmp=LL;
107         tmp=tmp->next;
108         while(tmp){
109             printf("%-8d%-8d ",tmp->start_byte,tmp->end_byte);
110             tmp=tmp->next;
111         }
112         printf("\n\n");
113     }
114 }

1
2 #include "LinkedList.h"
3
4 //Select first hole that is able to satisfy the process
5 void FirstFitInsert(Node *AllocLL, Node *FreePoolLL,Node *newNode){
6
7     Node *tmp=FreePoolLL;
8     Node *prev=tmp;
9     tmp=tmp->next;
10
11     //To determine whether a process can be allotted memory or not
12     int entry=0;
13     while(tmp){
14
15         if(tmp->size < newNode->size){
16             prev=tmp;
17             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
295 2.Process Exit \n 3.Display \n");
296         printf(" 4.Coalescing holes \n 0.Back \n Your
choice: ");
297         scanf("%d",&operation);
298
299         if(operation==1){
300             Node *process=(Node*)malloc(sizeof(Node));
301             initialise(process);
302             printf("\n Enter process ID: ");scanf(" %s",
process->status);
303             printf("\n Enter size required: ");scanf("%d",&
process->size);
304             if(FitOpt==1){
305                 FirstFitInsert(AllocLL,FreePoolLL,process);
306             }
307             else if(FitOpt==2){
308                 WorstFitInsert(AllocLL,FreePoolLL,process);
309             }
310             else if(FitOpt==3){
311                 BestFitInsert(AllocLL,FreePoolLL,process);
312             }
313             else if(FitOpt!=0){
314                 printf("\n Invalid algorithm. \n");
315             }
316             else;
317         }
318         else if(operation==2){
319             char *status=(char*)malloc(100);
320             printf("\n Enter process ID: ");scanf(" %s",
status);
321             DeallocMem(AllocLL,FreePoolLL,status);
322         }
323         else if(operation==3){
324             printf("\n Allocated Memory: \n");
325             displayMemory(AllocLL);
326             printf("\n Free Pool: \n");
327             displayMemory(FreePoolLL);
328             printf("\n Physical Memory: \n");
329             createPhysicalMem(AllocLL,FreePoolLL);
330         }
331         else if(operation==4){
332             holeJoin(FreePoolLL);
333             printf("\n Free Pool: \n");
334             displayMemory(FreePoolLL);
335         }
336         else if(operation!=0){
337             printf("\n Invalid operation. \n");
338         }
339         else;
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
```

```

57 3.Display
58 4.Coalescing holes
59 0.Back
60 Your choice: 1
61
62 Enter process ID: P4
63
64 Enter size required: 426
65
66 Process P4 cannot be allocated memory.
67
68 Choose operation:
69 1.Process Entry
70 2.Process Exit
71 3.Display
72 4.Coalescing holes
73 0.Back
74 Your choice: 3
75
76 Allocated Memory:
77 -----
78 |----- P1-----| |----- p3-----| |----- P2-----|
79 100          312 312          424 1100          1517
80
81
82 Free Pool:
83 -----
84 |----- H-----| |----- H-----| |----- H-----| |----- H-----|
85 0          100 424          600 600          800 800          1100
86 -----
87 |----- H-----|
88 1517          1700
89 Physical Memory:
90 -----
91 |----- H-----| |----- P1-----| |----- p3-----| |----- H-----|
92 0          100 100          312 312          424 424          600
93 -----
94 |----- H-----| |----- H-----| |----- P2-----| |----- H-----|
95 600          800 800          1100 1100          1517 1517          1700
96 Choose operation:
97 1.Process Entry
98 2.Process Exit
99 3.Display
100 4.Coalescing holes
101 0.Back
102 Your choice: 2
103
104 Enter process ID: P1
105
106 Choose operation:
107 1.Process Entry
108 2.Process Exit
109 3.Display
110 4.Coalescing holes
111 0.Back
112 Your choice: 2
113

```

```

114 Enter process ID: P2
115
116 Choose operation:
117 1.Process Entry
118 2.Process Exit
119 3.Display
120 4.Coalescing holes
121 0.Back
122 Your choice: 2
123
124 Enter process ID: p3
125
126 Choose operation:
127 1.Process Entry
128 2.Process Exit
129 3.Display
130 4.Coalescing holes
131 0.Back
132 Your choice: 3
133
134 Allocated Memory:
135
136 NULL
137
138 Free Pool:
139
140 |-----| |-----| |-----| |-----|
141 | H_ | | H_ | | H_ | | H_ |
142 | 0 100 100 312 312 424 424 600
143 |-----| |-----| |-----| |-----|
144 | H_ | | H_ | | H_ | | H_ |
145 | 600 800 800 1100 1100 1517 1517 1700
146 Physical Memory:
147
148 |-----| |-----| |-----| |-----|
149 | H_ | | H_ | | H_ | | H_ |
150 | 0 100 100 312 312 424 424 600
151 |-----| |-----| |-----| |-----|
152 | H_ | | H_ | | H_ | | H_ |
153 | 600 800 800 1100 1100 1517 1517 1700
154 Choose operation:
155 1.Process Entry
156 2.Process Exit
157 3.Display
158 4.Coalescing holes
159 0.Back
160 Your choice: 0
161
162 Choose the memory allocation algorithm:
163 1.First Fit
164 2.Worst Fit
165 3.Best Fit
166 0. Exit
167 Your choice: 2
168
169 Choose operation:
170 1.Process Entry
171 2.Process Exit
172 3.Display

```

```

171 4.Coalescing holes
172 0.Back
173 Your choice: 1
174
175 Enter process ID: P1
176
177 Enter size required: 212
178
179
180 Choose operation:
181 1.Process Entry
182 2.Process Exit
183 3.Display
184 4.Coalescing holes
185 0.Back
186 Your choice: 1
187
188 Enter process ID: P2
189
190 Enter size required: 417
191
192 Choose operation:
193 1.Process Entry
194 2.Process Exit
195 3.Display
196 4.Coalescing holes
197 0.Back
198 Your choice: 1
199
200 Enter process ID: P3
201
202 Enter size required: 112
203
204 Choose operation:
205 1.Process Entry
206 2.Process Exit
207 3.Display
208 4.Coalescing holes
209 0.Back
210 Your choice: 1
211
212 Enter process ID: P4
213
214 Enter size required: 426
215
216 Process P4 cannot be allocated memory.
217
218 Choose operation:
219 1.Process Entry
220 2.Process Exit
221 3.Display
222 4.Coalescing holes
223 0.Back
224 Your choice: 3
225
226 Allocated Memory:
227 -----

```

```

228 |----- P2-----| |----- P1-----| |----- P3-----|
229 100          517 1100          1312 1312          1424
230
231
232 Free Pool:
233 -----
234 |----- H-----| |----- H-----| |----- H-----| |----- H-----|
235 0          100 517          600 600          800 800          1100
236 -----
237 |----- H-----|
238 1424          1700
239
240 Physical Memory:
241 -----
242 |----- H-----| |----- P2-----| |----- H-----| |----- H-----|
243 0          100 100          517 517          600 600          800
244 -----
245 |----- H-----| |----- P1-----| |----- P3-----| |----- H-----|
246 800          1100 1100          1312 1312          1424 1424          1700
247
248
249 Choose operation:
250 1.Process Entry
251 2.Process Exit
252 3.Display
253 4.Coalescing holes
254 0.Back
255 Your choice: 2
256
257 Enter process ID: P1
258
259 Choose operation:
260 1.Process Entry
261 2.Process Exit
262 3.Display
263 4.Coalescing holes
264 0.Back
265 Your choice: 2
266
267 Enter process ID: P2
268
269 Choose operation:
270 1.Process Entry
271 2.Process Exit
272 3.Display
273 4.Coalescing holes
274 0.Back
275 Your choice: 2
276
277 Enter process ID: P3
278
279 Choose operation:
280 1.Process Entry
281 2.Process Exit
282 3.Display
283 4.Coalescing holes
284 0.Back

```

```

285 Your choice: 3
286
287 Allocated Memory:
288
289 NULL
290
291 Free Pool:
292
293 |-----| |-----| |-----| |-----|
294 |-----H-----| |-----H-----| |-----H-----| |-----H-----|
295 0          100 100          517 517          600 600          800
296 |-----| |-----| |-----| |-----|
297 |-----H-----| |-----H-----| |-----H-----| |-----H-----|
298 800          1100 1100          1312 1312          1424 1424          1700
299
299 Physical Memory:
300
301 |-----| |-----| |-----| |-----|
302 |-----H-----| |-----H-----| |-----H-----| |-----H-----|
303 0          100 100          517 517          600 600          800
304 |-----| |-----| |-----| |-----|
305 |-----H-----| |-----H-----| |-----H-----| |-----H-----|
306 800          1100 1100          1312 1312          1424 1424          1700
307
307 Choose operation:
308 1.Process Entry
309 2.Process Exit
310 3.Display
311 4.Coalescing holes
312 0.Back
313 Your choice: 0
314
315
316 Choose the memory allocation algorithm:
317 1.First Fit
318 2.Worst Fit
319 3.Best Fit
320 0. Exit
321 Your choice: 3
322
323 Choose operation:
324 1.Process Entry
325 2.Process Exit
326 3.Display
327 4.Coalescing holes
328 0.Back
329 Your choice: 1
330
331 Enter process ID: P1
332
333 Enter size required: 212
334
335 Choose operation:
336 1.Process Entry
337 2.Process Exit
338 3.Display
339 4.Coalescing holes
340 0.Back
341 Your choice: 1

```

```

342
343 Enter process ID: P2
344
345 Enter size required: 417
346
347 Choose operation:
348 1.Process Entry
349 2.Process Exit
350 3.Display
351 4.Coalescing holes
352 0.Back
353 Your choice: 1
354
355 Enter process ID: P3
356
357 Enter size required: 112
358
359 Choose operation:
360 1.Process Entry
361 2.Process Exit
362 3.Display
363 4.Coalescing holes
364 0.Back
365 Your choice: 1
366
367 Enter process ID: P4
368
369 Enter size required: 426
370
371 Choose operation:
372 1.Process Entry
373 2.Process Exit
374 3.Display
375 4.Coalescing holes
376 0.Back
377 Your choice: 3
378
379 Allocated Memory:
380 -----
381 |----- P2-----| |----- P3-----| |----- P1-----| |----- P4-----|
382 100          517 600          712 800          1012 1100          1526
383
384
385 Free Pool:
386 -----
387 |----- H-----| |----- H-----| |----- H-----| |----- H-----|
388 0          100 517          600 712          800 1012          1100
389
390 |----- H-----|
391 1526          1700
392
393 Physical Memory:
394 -----
395 |----- H-----| |----- P2-----| |----- H-----| |----- P3-----|
396 0          100 100          517 517          600 600          712
397 -----

```



```

398 |----- H-----| |----- P1-----| |----- H-----| |----- P4-----|
399 712 |----- H-----| 800 800 1012 1012 1100 1100 1526
      1526 1700
400
401
402 Choose operation:
403 1.Process Entry
404 2.Process Exit
405 3.Display
406 4.Coalescing holes
407 0.Back
408 Your choice: 2
409
410 Enter process ID: P1
411
412 Choose operation:
413 1.Process Entry
414 2.Process Exit
415 3.Display
416 4.Coalescing holes
417 0.Back
418 Your choice: 2
419
420 Enter process ID: P2
421
422 Choose operation:
423 1.Process Entry
424 2.Process Exit
425 3.Display
426 4.Coalescing holes
427 0.Back
428 Your choice: 2
429
430 Enter process ID: P3
431
432 Choose operation:
433 1.Process Entry
434 2.Process Exit
435 3.Display
436 4.Coalescing holes
437 0.Back
438 Your choice: 2
439
440 Enter process ID: P4
441
442 Choose operation:
443 1.Process Entry
444 2.Process Exit
445 3.Display
446 4.Coalescing holes
447 0.Back
448 Your choice: 3
449
450 Allocated Memory:
451
452 NULL

```

```

453
454   Free Pool:
455   -----
456   |----- H-----| |----- H-----| |----- H-----| |----- H-----|
457   0          100 100          517 517          600 600          712
458   -----
459   |----- H-----| |----- H-----| |----- H-----| |----- H-----|
460   712          800 800          1012 1012          1100 1100          1526
461   1526          1700
462
463   Physical Memory:
464   -----
465   |----- H-----| |----- H-----| |----- H-----| |----- H-----|
466   0          100 100          517 517          600 600          712
467   -----
468   |----- H-----| |----- P1-----| |----- H-----| |----- P4-----|
469   712          800 800          1012 1012          1100 1100          1526
470   1526          1700
471
472   Choose operation:
473   1.Process Entry
474   2.Process Exit
475   3.Display
476   4.Coalescing holes
477   0.Back
478   Your choice: 4
479
480   Free Pool:
481   -----
482   |----- H-----|
483   0          1700
484
485   Choose operation:
486   1.Process Entry
487   2.Process Exit
488   3.Display
489   4.Coalescing holes
490   0.Back
491   Your choice: 0
492
493   Choose the memory allocation algorithm:
494   1.First Fit
495   2.Worst Fit
496   3.Best Fit
497   0. Exit
498   Your choice: 0

```
