Submitted Files

Results | Code

memory.c

Download

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "oslabs.h"

struct MEMORY_BLOCK best_fit_allocate(int request_size, struct MEMORY_BLOCK memory_map[MAPMAX],int *map_cnt, int process_id) {
    struct MEMORY_BLOCK temp_memory_block, allocated_memory;
    allocated_memory.end_address = 0;
    allocated_memory.start_address = 0;
    allocated_memory.process_id = 0;
    allocated_memory.segment_size = 0;

    if (request_size == 0) {
        return allocated_memory;
    }

    bool match = false;
    int memory_map_index = 0 , best_fit_segment = 0;

    for (int i = 0; i<=*map_cnt; i++) {
        if ((memory_map[i].segment_size >= request_size) && (memory_map[i].process_id == 0)) {
            if (match == false) {
                memory_map_index = i;
                best_fit_segment = memory_map[i].segment_size;
                match = true;
            }
            else if (memory_map[i].segment_size < best_fit_segment) {
                memory_map_index = i;
                best_fit_segment = memory_map[i].segment_size;
            }
        }
    }

    if (match == true) {
        if (request_size < memory_map[memory_map_index].segment_size) {
            temp_memory_block = memory_map[memory_map_index];
            allocated_memory.start_address = memory_map[memory_map_index].start_address;
            allocated_memory.end_address = memory_map[memory_map_index].start_address + request_size - 1;
            allocated_memory.process_id = process_id;
            allocated_memory.segment_size = request_size;

            *map_cnt = *map_cnt + 1;
```

```
43              struct MEMORY_BLOCK temp_memory_block_2;
44
45              for (int i = memory_map_index; i <= *map_cnt; i++) {
46                  temp_memory_block_2 = memory_map[i+1];
47                  memory_map[i+1] = temp_memory_block;
48                  temp_memory_block = temp_memory_block_2;
49              }
50
51              memory_map[memory_map_index+1].start_address =
    allocated_memory.end_address + 1;
52              memory_map[memory_map_index+1].end_address =
    memory_map[memory_map_index].end_address;
53              memory_map[memory_map_index+1].process_id = 0;
54              memory_map[memory_map_index+1].segment_size =
    memory_map[memory_map_index].segment_size - allocated_memory.segment_size;
55
56              memory_map[memory_map_index] = allocated_memory;
57          }
58          else {
59              allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
60              allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
61              allocated_memory.process_id = process_id;
62              allocated_memory.segment_size = request_size;
63
64              memory_map[memory_map_index] = allocated_memory;
65          }
66      }
67      return allocated_memory;
68  }
69
70
71  struct MEMORY_BLOCK first_fit_allocate(int request_size, struct MEMORY_BLOCK
    memory_map[MAPMAX],int *map_cnt, int process_id)
72  {
73      struct MEMORY_BLOCK temp_memory_block, allocated_memory;
74      allocated_memory.end_address = 0;
75      allocated_memory.start_address = 0;
76      allocated_memory.process_id = 0;
77      allocated_memory.segment_size = 0;
78
79      if (request_size == 0) {
80          return allocated_memory;
81      }
82
83      bool match = false;
84      int memory_map_index = 0 , best_fit_segment = 0;
85
```

```
86      for (int i = 0; i<=*map_cnt; i++) {
87          if ((memory_map[i].segment_size >= request_size) &&
    (memory_map[i].process_id == 0)) {
88              if (match == false) {
89                  memory_map_index = i;
90                  best_fit_segment = memory_map[i].segment_size;
91                  match = true;
92                  break;
93              }
94              else if (memory_map[i].segment_size < best_fit_segment) {
95                  memory_map_index = i;
96                  best_fit_segment = memory_map[i].segment_size;
97              }
98          }
99      }
100
101     if (match == true) {
102         if (request_size < memory_map[memory_map_index].segment_size) {
103             temp_memory_block = memory_map[memory_map_index];
104             allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
105             allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
106             allocated_memory.process_id = process_id;
107             allocated_memory.segment_size = request_size;
108
109             *map_cnt = *map_cnt + 1;
110             struct MEMORY_BLOCK temp_memory_block_2;
111
112             for (int i = memory_map_index; i <= *map_cnt; i++) {
113                 temp_memory_block_2 = memory_map[i+1];
114                 memory_map[i+1] = temp_memory_block;
115                 temp_memory_block = temp_memory_block_2;
116             }
117
118             memory_map[memory_map_index+1].start_address =
    allocated_memory.end_address + 1;
119             memory_map[memory_map_index+1].end_address =
    memory_map[memory_map_index].end_address;
120             memory_map[memory_map_index+1].process_id = 0;
121             memory_map[memory_map_index+1].segment_size =
    memory_map[memory_map_index].segment_size - allocated_memory.segment_size;
122
123             memory_map[memory_map_index] = allocated_memory;
124         }
125         else {
126             allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
```

```
127             allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
128             allocated_memory.process_id = process_id;
129             allocated_memory.segment_size = request_size;
130 
131             memory_map[memory_map_index] = allocated_memory;
132         }
133     }
134     return allocated_memory;
135 }
136 
137 
138 
139 
140 
141 
142 struct MEMORY_BLOCK worst_fit_allocate(int request_size, struct MEMORY_BLOCK
    memory_map[MAPMAX],int *map_cnt, int process_id)
143 {
144     struct MEMORY_BLOCK temp_memory_block, allocated_memory;
145     allocated_memory.end_address = 0;
146     allocated_memory.start_address = 0;
147     allocated_memory.process_id = 0;
148     allocated_memory.segment_size = 0;
149 
150     if (request_size == 0) {
151         return allocated_memory;
152     }
153 
154     bool match = false;
155     int memory_map_index = 0 , best_fit_segment = 0;
156 
157     for (int i = 0; i<=*map_cnt; i++) {
158         if ((memory_map[i].segment_size >= request_size) &&
    (memory_map[i].process_id == 0)) {
159             if (match == false) {
160                 memory_map_index = i;
161                 best_fit_segment = memory_map[i].segment_size;
162                 match = true;
163             }
164             else if (memory_map[i].segment_size > best_fit_segment) {
165                 memory_map_index = i;
166                 best_fit_segment = memory_map[i].segment_size;
167             }
168         }
169     }
170 
171     if (match == true) {
172         if (request_size < memory_map[memory_map_index].segment_size) {
```

```
173             temp_memory_block = memory_map[memory_map_index];
174             allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
175             allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
176             allocated_memory.process_id = process_id;
177             allocated_memory.segment_size = request_size;
178 
179             *map_cnt = *map_cnt + 1;
180             struct MEMORY_BLOCK temp_memory_block_2;
181 
182             for (int i = memory_map_index; i <= *map_cnt; i++) {
183                 temp_memory_block_2 = memory_map[i+1];
184                 memory_map[i+1] = temp_memory_block;
185                 temp_memory_block = temp_memory_block_2;
186             }
187 
188             memory_map[memory_map_index+1].start_address =
    allocated_memory.end_address + 1;
189             memory_map[memory_map_index+1].end_address =
    memory_map[memory_map_index].end_address;
190             memory_map[memory_map_index+1].process_id = 0;
191             memory_map[memory_map_index+1].segment_size =
    memory_map[memory_map_index].segment_size - allocated_memory.segment_size;
192 
193             memory_map[memory_map_index] = allocated_memory;
194         }
195         else {
196             allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
197             allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
198             allocated_memory.process_id = process_id;
199             allocated_memory.segment_size = request_size;
200 
201             memory_map[memory_map_index] = allocated_memory;
202         }
203     }
204     return allocated_memory;
205 }
206 
207 
208 
209 struct MEMORY_BLOCK next_fit_allocate(int request_size, struct MEMORY_BLOCK
    memory_map[MAPMAX],int *map_cnt, int process_id, int last_address) {
210 
211     struct MEMORY_BLOCK temp_memory_block, allocated_memory;
212     allocated_memory.end_address = 0;
213     allocated_memory.start_address = 0;
```

```
214     allocated_memory.process_id = 0;
215     allocated_memory.segment_size = 0;
216 
217     if (request_size == 0) {
218         return allocated_memory;
219     }
220 
221     bool match = false;
222     int memory_map_index = 0 , best_fit_segment = 0;
223 
224     for (int i = 0; i<=*map_cnt; i++) {
225         if ((memory_map[i].segment_size >= request_size) &&
    (memory_map[i].process_id == 0) && (memory_map[i].start_address >=
    last_address)) {
226             if (match == false) {
227                 memory_map_index = i;
228                 best_fit_segment = memory_map[i].segment_size;
229                 match = true;
230                 break;
231             }
232             else if (memory_map[i].segment_size < best_fit_segment) {
233                 memory_map_index = i;
234                 best_fit_segment = memory_map[i].segment_size;
235             }
236         }
237     }
238 
239     if (match == true) {
240         if (request_size < memory_map[memory_map_index].segment_size) {
241             temp_memory_block = memory_map[memory_map_index];
242             allocated_memory.start_address =
    memory_map[memory_map_index].start_address;
243             allocated_memory.end_address =
    memory_map[memory_map_index].start_address + request_size - 1;
244             allocated_memory.process_id = process_id;
245             allocated_memory.segment_size = request_size;
246 
247             *map_cnt = *map_cnt + 1;
248             struct MEMORY_BLOCK temp_memory_block_2;
249 
250             for (int i = memory_map_index; i <= *map_cnt; i++) {
251                 temp_memory_block_2 = memory_map[i+1];
252                 memory_map[i+1] = temp_memory_block;
253                 temp_memory_block = temp_memory_block_2;
254             }
255 
256             memory_map[memory_map_index+1].start_address =
    allocated_memory.end_address + 1;
```

```
257             memory_map[memory_map_index+1].end_address =
memory_map[memory_map_index].end_address;
258             memory_map[memory_map_index+1].process_id = 0;
259             memory_map[memory_map_index+1].segment_size =
memory_map[memory_map_index].segment_size - allocated_memory.segment_size;
260 
261             memory_map[memory_map_index] = allocated_memory;
262         }
263         else {
264             allocated_memory.start_address =
memory_map[memory_map_index].start_address;
265             allocated_memory.end_address =
memory_map[memory_map_index].start_address + request_size - 1;
266             allocated_memory.process_id = process_id;
267             allocated_memory.segment_size = request_size;
268 
269             memory_map[memory_map_index] = allocated_memory;
270         }
271     }
272     return allocated_memory;
273 }
274 
275 
276 
277 void release_memory(struct MEMORY_BLOCK freed_block, struct MEMORY_BLOCK
memory_map[MAPMAX],int *map_cnt)
278 {
279    bool flag = false;
280 if ((*map_cnt == 1) && (memory_map[0].end_address == 0) &&
(memory_map[0].start_address == 0) && (memory_map[0].process_id == 0) &&
(memory_map[0].segment_size == 0))
281     return;
282 else
283 {
284     for (int i = 0; i < *map_cnt; i++)
285     {
286         if((freed_block.start_address == memory_map[i].start_address) &&
(freed_block.end_address == memory_map[i].end_address) &&
(freed_block.process_id == memory_map[i].process_id)) {
287         memory_map[i].process_id = 0;
288         if (i > 0)
289         {
290             if (memory_map[i-1].process_id == 0)
291             {
292                 memory_map[i-1].end_address = freed_block.end_address;
293                 memory_map[i-1].segment_size = memory_map[i-1].segment_size +
freed_block.segment_size;
294                 for (int index = i; index <= *map_cnt; index++)
295                 {
```

```
296                     memory_map[index] = memory_map[index + 1];
297                 }
298                 *map_cnt = *map_cnt - 1;
299                 flag = true;
300             }
301         }
302         if (i < *map_cnt-1)
303         {
304             if (flag == false)
305             {
306                 i = i+1;
307             }
308             if (memory_map[i].process_id == 0)
309             {
310                 memory_map[i].start_address = memory_map[i-1].start_address;
311                 memory_map[i].segment_size = memory_map[i].end_address -
    memory_map[i].start_address+1;
312                 for (int index = i; index <= *map_cnt; index++)
313                 {
314                     memory_map[index-1] = memory_map[index];
315                 }
316                 *map_cnt = *map_cnt - 1;
317             }
318         }
319         break;
320         }
321     }
322 }
323 }
```

Memory Lab

Graded

Select each question to review feedback and grading details.

Student

Jahan Amrin

Total Points

5 / 5 pts

Autograder Score
5.0 / 5.0

Passed Tests

best_fit_allocate (1/1)
first_fit_allocate (1/1)
next_fit_allocate (1/1)
release_memory (1/1)
worst_fit_allocate (1/1)