

Project7 实验报告

余北辰 519030910245

1 实验概述

1.1 实验名称

Contiguous Memory Allocation

1.2 实验内容

1. 实现动态内存分配的模拟
2. 实现对 RQ、RL、C、STAT、X 五种指令的读取，并分别实现内存申请、内存释放、内存紧缩、打印现有状态和结束模拟的功能
3. 内存申请时根据指令 F、B、W 分别进行First-fit,Best-fit和Worst-fit分配

2 实验环境

- Ubuntu 18.04.5 LTS
- Linux version 5.4.0-72-generic
- VirtualBox 6.1.18

3 实验过程与结果展示

使用链表维护内存空间。

3.1 变量定义

定义 Hole 这一结构，begin、end、size 分别代表某一内存块的起始位置、终止位置和内存块的大小；used 代表这一内存块是已然被分配给某一进程的，还是尚未分配的；name 指针指向所分配的进程的名称，若未分配则指向空字符串。

```
1 int total_memory;  
2
```

```

3  typedef struct Hole
4  {
5      int begin;
6      int end;
7      int size;
8      int used;
9      char *name;
10 } Hole;
11
12 typedef struct Node
13 {
14     struct Node *next;
15     struct Hole hole;
16 } Node;
17
18 Node *head;

```

3.2 内存申请

First-fit

从头节点开始，一直向后寻找，直至找到第一个未被使用，且大小足够的 **Hole**。找到合适的 **Hole** 之后，将这个 **Hole** 分裂，靠前的部分修改为已分配状态，靠后的部分仍为未分配状态。

注意当 **Hole** 的大小刚好合适的时候，不需要分裂该 **Hole**，只需修改该 **Hole** 的状态即可。

First-fit() 实现的代码如下：

```

1  void first_fit(char *name, int size)
2  {
3      Node *p = head;
4      while (1)
5      {
6          if (p->hole.used == 0 && p->hole.size > size)
7          {
8              Node *q = (Node *)malloc(sizeof(Node));
9              p->hole.used = 1;
10             q->hole.used = 0;
11             p->hole.name = name;
12             q->hole.name = "";
13             q->hole.end = p->hole.end;
14             p->hole.end = p->hole.begin + size;
15             p->hole.size = size;

```

```

16         q->hole.begin = p->hole.end;
17         q->hole.size = q->hole.end - q->hole.begin;
18         q->next = p->next;
19         p->next = q;
20         break;
21     }
22     else if (p->hole.used == 0 && p->hole.size == size)
23     {
24         p->hole.used = 1;
25         p->hole.name = name;
26         break;
27     }
28     else
29     {
30         if (p->next != NULL)
31         {
32             p = p->next;
33         }
34         else
35         {
36             fprintf(stderr, "Error: Failed! No suitable space!\n");
37             break;
38         }
39     }
40 }
41 }

```

Best-fit

Best-fit和first-fit的区别在于，要找到size最小的未分配的 **Hole**。

只需先遍历链表，找到size最小的未分配的 **Hole**，再类似于first-fit的操作进行分配即可。

Best-fit() 实现的代码如下：

```

1 void best_fit(char *name, int size)
2 {
3     Node *p = head;
4     int minsize = __INT_MAX__;
5     Node *minnode = NULL;
6     while (1)
7     {
8         if (p->hole.used == 0 && p->hole.size >= size)
9         {

```

```

10     if (p->hole.size < minsize)
11     {
12         minsize = p->hole.size;
13         minnode = p;
14     }
15 }
16 if (p->next != NULL)
17 {
18     p = p->next;
19 }
20 else
21 {
22     if (minsize == __INT_MAX__)
23         fprintf(stderr, "Error: Failed! No suitable space!\n");
24     break;
25 }
26 }
27 if (minsize != __INT_MAX__)
28 {
29     p = minnode;
30     if (minsize == size)
31     {
32         p->hole.used = 1;
33         p->hole.name = name;
34     }
35     else
36     {
37         Node *q = (Node *)malloc(sizeof(Node));
38         p->hole.used = 1;
39         q->hole.used = 0;
40         p->hole.name = name;
41         q->hole.name = "";
42         q->hole.end = p->hole.end;
43         p->hole.end = p->hole.begin + size;
44         p->hole.size = size;
45         q->hole.begin = p->hole.end;
46         q->hole.size = q->hole.end - q->hole.begin;
47         q->next = p->next;
48         p->next = q;
49     }
50 }
51 }

```

Worst-fit

Worst-fit和Best-fit很相似，只是Best-fit寻找的是size最小的未分配的 **Hole**，而Worst-fit寻找的则是size最大的。

Worst-fit() 的实现代码如下：

```
1 void worst_fit(char *name, int size)
2 {
3     Node *p = head;
4     int maxsize = -1;
5     Node *maxnode = NULL;
6     while (1)
7     {
8         if (p->hole.used == 0 && p->hole.size >= size)
9         {
10             if (p->hole.size > maxsize)
11             {
12                 maxsize = p->hole.size;
13                 maxnode = p;
14             }
15         }
16
17         if (p->next != NULL)
18         {
19             p = p->next;
20         }
21         else
22         {
23             if (maxsize == -1)
24                 fprintf(stderr, "Error: Failed! No suitable space!\n");
25             break;
26         }
27     }
28     if (maxsize != -1)
29     {
30         p = maxnode;
31         if (maxsize == size)
32         {
33             p->hole.used = 1;
34             p->hole.name = name;
35         }
36         else
37         {
```

```

38     Node *q = (Node *)malloc(sizeof(Node));
39     p->hole.used = 1;
40     q->hole.used = 0;
41     p->hole.name = name;
42     q->hole.name = "";
43     q->hole.end = p->hole.end;
44     p->hole.end = p->hole.begin + size;
45     p->hole.size = size;
46     q->hole.begin = p->hole.end;
47     q->hole.size = q->hole.end - q->hole.begin;
48     q->next = p->next;
49     p->next = q;
50 }
51 }
52 }

```

3.3 内存释放

相比内存申请而言，内存释放的操作需要更加复杂一些。因为内存申请时，只要把未分配的 **Hole** 分裂即可；而内存释放时，在修改对应 **Hole** 的状态后，还需要分多种情况考虑，让相邻的未分配的 **Hole** 合并起来。

而且由于涉及到修改链表各个 **Node** 之间的连接关系，需要使用双指针。这也意味着需要考虑释放头结点中 **Hole** 的特殊情况。

release() 的具体实现过程代码如下所示：

```

1 void release(char *name)
2 {
3     Node *p = head->next;
4     Node *q = head;
5
6     while (1)
7     {
8         if (q == head && strcmp(q->hole.name, name) == 0)
9         {
10             if (p == NULL)
11             {
12                 q->hole.begin = 0;
13                 q->hole.end = total_memory;
14                 q->hole.used = 0;
15                 q->hole.size = total_memory;
16                 q->hole.name = "";
17                 break;
18             }

```

```

19     else if (p->hole.used == 1)
20     {
21         q->hole.used = 0;
22         q->hole.name = "";
23         break;
24     }
25     else
26     {
27         q->hole.end = p->hole.end;
28         q->hole.size = q->hole.end;
29         q->hole.name = "";
30         q->hole.used = 0;
31         q->next = p->next;
32         free(p);
33     }
34 }
35
36 if (p == NULL)
37 {
38     fprintf(stderr, "Error: Failed! No that process!\n");
39     break;
40 }
41
42 if (strcmp(p->hole.name, name) == 0)
43 {
44     Node *r = p->next;
45     if (q->hole.used == 0)
46     {
47         if (r == NULL)
48         {
49             q->hole.end = p->hole.end;
50             q->hole.size = q->hole.end - q->hole.begin;
51             q->next = p->next;
52             free(p);
53         }
54         else if (r->hole.used == 1)
55         {
56             q->hole.end = p->hole.end;
57             q->hole.size = q->hole.end - q->hole.begin;
58             q->next = p->next;
59             free(p);
60         }
61         else

```

```
62         {
63             q->hole.end = r->hole.end;
64             q->hole.size = q->hole.end - q->hole.begin;
65             q->next = r->next;
66             free(p);
67             free(r);
68         }
69     }
70     else
71     {
72         if (r == NULL)
73         {
74             p->hole.used = 0;
75             p->hole.name = "";
76         }
77         else if (r->hole.used == 1)
78         {
79             p->hole.used = 0;
80             p->hole.name = "";
81         }
82         else
83         {
84             p->hole.used = 0;
85             p->hole.name = "";
86             p->hole.end = r->hole.end;
87             p->hole.size = p->hole.end - p->hole.begin;
88             p->next = r->next;
89             free(r);
90         }
91     }
92     break;
93 }
94 else
95 {
96     if (p->next != NULL)
97     {
98         q = q->next;
99         p = p->next;
100    }
101    else
102    {
103        fprintf(stderr, "Error: Failed! No that process!\n");
104        break;
```



```
105     }
106     }
107     }
108 }
```

3.4 内存紧缩

内存紧缩的过程相对简单一些，只需要从头节点开始，凡是遇到未分配的 **Hole** 就令其前后 **Node** 相连、将其所在的 **Node** 删去即可。

但注意要维护 **begin** 和 **end** 的值。

同样使用了双指针，要特殊考虑头节点的情况。

对于最后一个 **Hole**，也要特殊考虑。

compact() 的具体实现代码如下：

```
1 void compact()
2 {
3     int offset = 0;
4     Node *q = head;
5     Node *p = head->next;
6     if (head->hole.used == 0)
7     {
8         offset += head->hole.size;
9     }
10    while (1)
11    {
12        if (p->next == NULL)
13        {
14            if (p->hole.used == 0)
15            {
16                p->hole.begin -= offset;
17                p->hole.size += offset;
18            }
19            else
20            {
21                p->hole.begin -= offset;
22                p->hole.end -= offset;
23                Node *end = (Node *)malloc(sizeof(Node));
24                end->hole.begin = p->hole.end;
25                end->hole.end = total_memory;
26                end->hole.size = end->hole.end - end->hole.begin;
27                end->hole.used = 0;
28                end->hole.name = "";
29                p->next = end;
```

```

30         end->next = NULL;
31     }
32     break;
33 }
34 if (p->hole.used == 0)
35 {
36     offset += p->hole.size;
37     q->next = p->next;
38     Node *freeNode = p;
39     p = p->next;
40     free(freeNode);
41 }
42 else
43 {
44     p->hole.begin -= offset;
45     p->hole.end -= offset;
46     p = p->next;
47     q = q->next;
48 }
49 }
50 if (head->hole.used == 0 && head->next != NULL)
51 {
52     Node *freeNode = head;
53     head = head->next;
54     free(freeNode);
55 }
56 }

```

3.5 其他部分

打印当前状态

代码实现如下：

```

1 void status_report()
2 {
3     Node *p = head;
4     while (1)
5     {
6         printf("Addresses ");
7         if (p == head)
8             printf("[%d:%d] ", p->hole.begin, p->hole.end);
9         else
10            printf("[%d,%d] ", p->hole.begin + 1, p->hole.end);

```

```

11
12     if (p->hole.used == 0)
13         printf("Unused \n");
14     else
15         printf("Process %s\n", p->hole.name);
16
17     if (p->next != NULL)
18         p = p->next;
19     else
20         break;
21 }
22 }

```

命令识别与选择

在 `main()` 函数实现。

代码如下：

```

1  int main(int argc, char **argv)
2  {
3      if (argc != 2)
4      {
5          fprintf(stderr, "Error: Please input right arguments!\n");
6          return -1;
7      }
8      head = (Node *)malloc(sizeof(Node));
9      head->next = NULL;
10
11     total_memory = atoi(argv[1]);
12     head->hole.begin = 0;
13     head->hole.end = total_memory;
14     head->hole.size = total_memory;
15     head->hole.used = 0;
16     head->hole.name = "";
17     char buffer[20];
18
19     while (1)
20     {
21         printf("allocator> ");
22         scanf("%s", buffer);
23         if (strcmp(buffer, "RQ") == 0)
24         {
25             Node *q = (Node *)malloc(sizeof(Node));

```

```
26
27     char *name_of_process = (char *)malloc(10 * sizeof(char));
28     scanf("%s", name_of_process);
29     int size_of_process;
30     scanf("%d", &size_of_process);
31     char *type = (char *)malloc(3 * sizeof(char));
32     scanf("%s", type);
33
34     if (strcmp(type, "F") == 0)
35     {
36         first_fit(name_of_process, size_of_process);
37     }
38     else if (strcmp(type, "B") == 0)
39     {
40         best_fit(name_of_process, size_of_process);
41     }
42     else if (strcmp(type, "W") == 0)
43     {
44         worst_fit(name_of_process, size_of_process);
45     }
46 }
47
48 else if (strcmp(buffer, "RL") == 0)
49 {
50     char *name_of_process = (char *)malloc(10 * sizeof(char));
51     scanf("%s", name_of_process);
52     release(name_of_process);
53     free(name_of_process);
54 }
55 else if (strcmp(buffer, "C") == 0)
56 {
57     compact();
58 }
59 else if (strcmp(buffer, "STAT") == 0)
60 {
61     status_report();
62 }
63 else if (strcmp(buffer, "X") == 0)
64 {
65     break;
66 }
67 else
68 {
```

```

69         fprintf(stderr, "Error: Please input right commands!\n");
70     }
71 }
72
73     return 0;
74 }

```

3.6 测试结果

测试First-fit:

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project7
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ ./allocator 1048576
allocator>RQ P0 40000 W
allocator>RQ P1 50000 B
allocator>RQ P2 30000 F
allocator>RQ P3 40000 W
allocator>RQ P4 90000 B
allocator>RL P1
allocator>RL P3
allocator>RQ P5 35000 F
allocator>STAT
Addresses [0:40000] Process P0
Addresses [40001,75000] Process P5
Addresses [75001,90000] Unused
Addresses [90001,120000] Process P2
Addresses [120001,160000] Unused
Addresses [160001,250000] Process P4
Addresses [250001,1048576] Unused
allocator>X
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ |

```

测试Best-fit:

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project7
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ ./allocator 1048576
allocator>RQ P0 40000 W
allocator>RQ P1 50000 B
allocator>RQ P2 30000 F
allocator>RQ P3 40000 W
allocator>RQ P4 90000 B
allocator>RL P1
allocator>RL P3
allocator>RQ P5 35000 B
allocator>STAT
Addresses [0:40000] Process P0
Addresses [40001,90000] Unused
Addresses [90001,120000] Process P2
Addresses [120001,155000] Process P5
Addresses [155001,160000] Unused
Addresses [160001,250000] Process P4
Addresses [250001,1048576] Unused
allocator>X
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ |

```

测试Worst-fit:

```
polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project7
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ ./allocator 1048576
allocator>RQ P0 40000 W
allocator>RQ P1 50000 B
allocator>RQ P2 30000 F
allocator>RQ P3 40000 W
allocator>RQ P4 90000 B
allocator>RL P1
allocator>RL P3
allocator>RQ P5 35000 W
allocator>STAT
Addresses [0:40000] Process P0
Addresses [40001,90000] Unused
Addresses [90001,120000] Process P2
Addresses [120001,160000] Unused
Addresses [160001,250000] Process P4
Addresses [250001,285000] Process P5
Addresses [285001,1048576] Unused
allocator>X
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ |
```

测试compact:

```
polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project7
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ ./allocator 1048576
allocator>RQ P0 40000 W
allocator>RQ P1 50000 B
allocator>RQ P2 30000 F
allocator>RQ P3 40000 W
allocator>RQ P4 90000 B
allocator>RL P0
allocator>RL P2
allocator>RL P4
allocator>C
allocator>STAT
Addresses [0:50000] Process P1
Addresses [50001,90000] Process P3
Addresses [90001,1048576] Unused
allocator>X
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project7$ |
```

4 实验总结

1. 注意在内存释放和内存紧缩时，要对相应的 Node 调用 `free()` 函数，以避免内存泄漏。
2. 注意要特殊考虑头尾节点。

5 实验参考资料

- 实验参考书籍：Operating System Concept, 10th edition
- 实验源代码网址：<https://github.com/greggagne/osc10e>

