# Project 7: Contiguous Memory Allocation

（一）问题分析

需要实现一个连续内存分配模型，支持 4 种操作：分配内存，释放内存，整合内存，查看内存使用情况。其中，分配内存可选三种策略：首次适应、最优适应或最差适应。

- 首次适应：分配首个足够大的孔。查找可以从头开始，也可以从上次首次适应结束时开始。一旦找到足够大的空闲孔，就可以停止。
- 最优适应：分配最小的足够大的孔。应查找整个列表，除非列表按大小排序。这种方法可以产生最小剩余孔。
- 最差适应：分配最大的孔。同样，应查找整个列表，除非列表按大小排序。这种方法可以产生最大剩余孔，该孔可能比最优适应产生的较小剩余孔更为适用。

（二）实现细节

一、数据结构

```
 7 typedef struct Hole
 8 {
 9         int id;
10         int start;
11         int end;
12         int size;
13
14         struct Hole* next;
15 } Hole;
```

用链表来模拟连续内存空间，结点有五个属性：

1、id：记录结点类型——id = -1 表示该结点为孔，id = x 表示该结点为进程 Px。

2、start：结点在内存中的开始位置。

3、end：结点在内存中的结束位置。

4、size：结点在内存中占多少空间。

5、*next：指向下一个结点的指针。

二、全局变量

1、*memory：指向连续内存链表头结点的指针。

2、MAX：连续内存空间的大小。

三、函数

1、request()

（1）检查传入的参数是否合法：size 是否为正，id 是否非负。非法则在控制台输出错误信息，退出函数。

（2）根据 type 调用对应的策略的函数。如果没有对应的策略，则在控制台输出错误信息。

（3）退出函数。

2、first_fit()

（1）用指针 p 来遍历链表。

（2）如果 p 指向的结点为孔：

①如果该孔的空间大小等于进程申请的空间的大小，则直接整块分配，即根据申请空间的进程的 id 更新该孔的 id。退出函数。

②如果该孔的空间大小大于进程申请的空间的大小，则逻辑上需要将该孔一分为二，前一部分根据申请空间的进程修改各属性，后一部分依据剩余空间修改属性。实现上，p 指向的结点依据申请空间的进程修改属性，还需要新建一个结点，依据剩余空间初始化属性。最后将该新结点插入到 p 指向的结点之后。退出函数。

③如果该孔的空间大小小于进程申请的空间的大小，则继续遍历。

（3）如果 p 指向的结点不为孔，则继续遍历。

（4）遍历结束后，仍然没有退出函数，说明没有符合条件的空间可供分配，在控制台输出错误信息，退出函数。

3、best_fit()

（1）用指针 p 来遍历链表，指针 best 来记录最优孔。

（2）如果 p 指向的结点为孔：

①如果该孔为遍历过程中遇到的第一个孔，则用 best 记录它；

②如果该孔的空间大于等于进程申请的空间，且小于目前 best 指向的孔的空间，则用 best 记录它；

③如果该孔的空间小于进程申请的空间，则继续遍历。

（3）如果 p 指向的结点不为孔，则继续遍历。

（4）遍历结束后，如果 best 不为孔，明没有符合条件的空间可供分配，在控制台输出提示信息，退出函数。如果 best 为孔：

①如果该孔的空间大小等于进程申请的空间的大小，同理 first_fit()（2）①。

②如果该孔的空间大小大于进程申请的空间的大小，同理 first_fit()（2）②。

4、worst_fit()

（1）用指针 p 来遍历链表，指针 worst 来记录最大孔。

（2）如果 p 指向的结点为孔：

①如果该孔为遍历过程中遇到的第一个孔，则用 worst 记录它。

②如果该孔的空间大于等于进程申请的空间，且大于目前 worst 指向的孔的空间，则用 worst 记录它。

③如果该孔的空间小于进程申请的空间，则继续遍历。

（3）如果 p 指向的结点不为孔，则继续遍历。

（4）遍历结束后，如果 worst 不为孔，明没有符合条件的空间可供分配，在控制台输出提示信息，退出函数。如果 worst 为孔：

①如果该孔的空间大小等于进程申请的空间的大小，同理 first_fit()（2）①。

②如果该孔的空间大小大于进程申请的空间的大小，同理 first_fit()（2）②。

5、release()

（1）检查传入的参数是否合法：id 是否非负。

（2）首先处理头结点为目标结点的情况，对于头结点存在三种情况：

①head->NULL：直接释放空间，即修改 id 为 -1，退出函数。

②head->process：直接释放空间，即修改 id 为 -1，退出函数。

③head->hole:逻辑上释放空间后还需要将空间合并。实现上,求出合并后有多少空间，据此修改 head 指向的结点的属性。还需要释放掉后一个结点。退出函数。

（3）如果头结点不是目标结点，则用指针 p 遍历链表，指针 prev 记录当前结点的前一个结点。
（4）p 为目标结点有六种情况：
①process->target->NULL：直接释放空间，即修改 id 为 -1，退出函数。
②process ->target->process：直接释放空间，即修改 id 为 -1，退出函数。
③process ->target->hole：逻辑上释放空间后还需要将空间合并。实现上，求出合并后有多少空间，据此修改 target 指向的结点的属性，然后释放掉后一个结点。退出函数。
④hole->target->NULL：逻辑上释放空间后还需要将空间合并。实现上，求出合并后有多少空间，据此修改 prev 指向的结点的属性，然后释放掉 target 指向的结点。退出函数。
⑤hole->target->process：逻辑上释放空间后还需要将空间合并。实现上，求出合并后有多少空间，据此修改 prev 指向的结点的属性，还要将 prev 的 next 指针指向 process，然后释放掉 target 指向的结点。退出函数。
⑥hole->target->hole：逻辑上释放空间后还需要将空间合并。实现上，求出合并后有多少空间，据此修改 prev 指向的结点的属性，还要将 prev 的 next 指针指向 target 后的第二个结点，然后释放掉 target 指向的结点和 target 后的第一个结点。退出函数。
（5）遍历结束后，仍然没有退出函数，说明没有符合条件的进程，在控制台输出错误信息，退出函数。

6、compact()
（1）如果内存里没有进程或全部为一个进程，则不需要整合。退出函数。
（2）用指针 p 遍历链表，指针 prev 记录上一个为 process 的结点，初始化为空。用 unused_space 记录剩余空间总和。
（3）如果 p 指向的结点为孔，则更新 unused_space，释放掉这个结点。
（4）如果 p 指向的结点为进程：
①如果 prev 为空即这是第一个进程，则用 prev 记录这个结点。
②如果这是下一个进程，则将 p 指向的结点连接到 prev 指向的结点之后，修改 p 指向的结点的属性，然后让 prev 记录 p 指向的结点，p 继续遍历。
（5）遍历结束后，内存中所有的进程对应的结点已相邻，所有孔对应的结点都被释放。新建一个结点，根据 prev 和 unused_space 修改其属性，然后把它连接到链表末尾。
（6）退出函数。

7、stat()
（1）用指针 p 遍历链表。
（2）根据各结点属性，输出相应信息。
（3）退出函数。

四、主函数
1、从命令行参数获取 MAX。
2、初始化 memory。
3、仿照 project2-1：Unix Shell 写命令行“allocator>”主循环代码框架。
4、input[]存储输入的指令。
5、RQ 指令
处理 input[]得到进程编号 id，进程申请的空间 size，采用的分配策略 type，调用函数 request()。

6、RL 指令

处理 input[]得到进程编号 id，调用函数 release()。

7、C 指令

调用函数 compact()。

8、STAT 指令

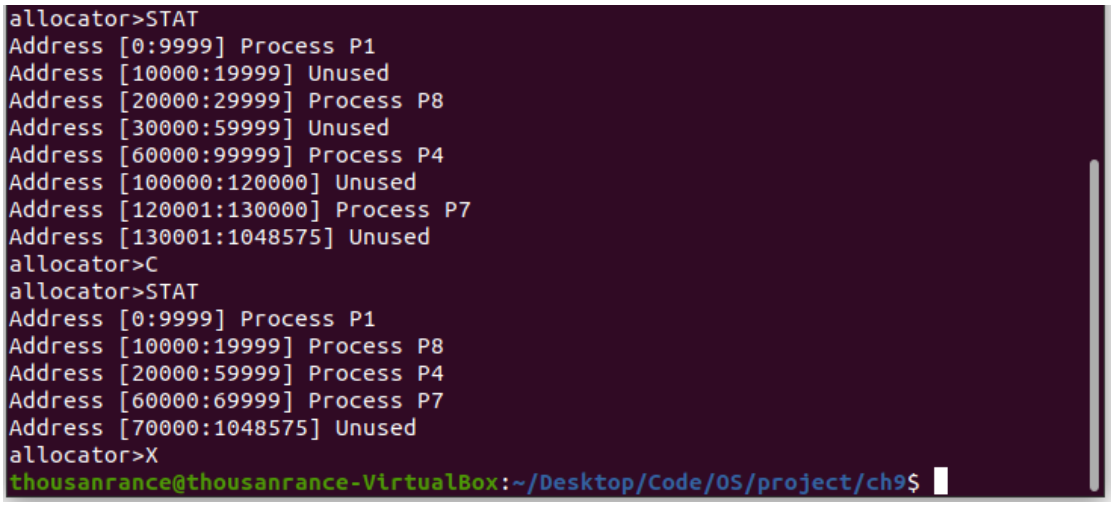
调用函数 stat()。

9、X 指令

退出 allocator 命令行，即程序终止

（三）运行结果

```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch9$ gcc -o allo
cator allocator.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch9$ ./allocator
 1048576
allocator>RQ P1 10000 B
allocator>RQ P2 20000 B
allocator>RQ P3 30000 B
allocator>RQ P4 40000 B
allocator>STAT
Address [0:9999] Process P1
Address [10000:29999] Process P2
Address [30000:59999] Process P3
Address [60000:99999] Process P4
Address [100000:1048575] Unused
allocator>RL P2
allocator>RQ P5 20001 F
allocator>STAT
Address [0:9999] Process P1
Address [10000:29999] Unused
Address [30000:59999] Process P3
Address [60000:99999] Process P4
Address [100000:120000] Process P5
Address [120001:1048575] Unused
allocator>RQ P6 10000 B
allocator>RQ P7 10000 W
allocator>RQ P8 10000 F
allocator>STAT
Address [0:9999] Process P1
Address [10000:19999] Process P6
Address [20000:29999] Process P8
Address [30000:59999] Process P3
Address [60000:99999] Process P4
Address [100000:120000] Process P5
Address [120001:130000] Process P7
Address [130001:1048575] Unused
allocator>RL P6
allocator>RL P3
allocator>RL P5
```


allocator>STAT
Address [0:9999] Process P1
Address [10000:19999] Unused
Address [20000:29999] Process P8
Address [30000:59999] Unused
Address [60000:99999] Process P4
Address [100000:120000] Unused
Address [120001:130000] Process P7
Address [130001:1048575] Unused
allocator>C
allocator>STAT
Address [0:9999] Process P1
Address [10000:19999] Process P8
Address [20000:59999] Process P4
Address [60000:69999] Process P7
Address [70000:1048575] Unused
allocator>X
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch9$