实验报告

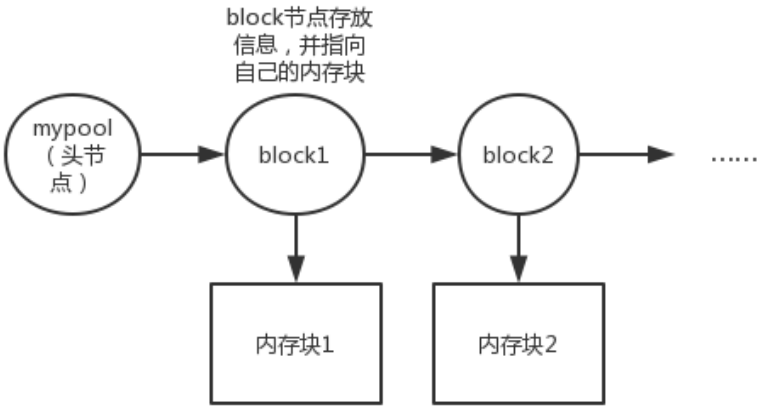
实验名称	实验三：C 内存管理程序设计

1. 攻击面分析：
   1) 函数 int initPool(size_t size) ：
      - 内存池申请内存失败会导致后续操作均出现问题；
      - 内存块是逐块申请内存的，一旦有申请失败的，则内存池也不可用，前面申请的内存仍然在，若不释放，将导致内存泄漏；
      - 释放完内存，若不将指针置为空，将导致也野指针的存在，后续操作导致内存访问异常。
   2) 函数 char* allocBlock()：
      - 内存池未初始化成功，无法分配；
      - 如果内存池需要扩展，使用 realloc 函数会导致无法回收内存块。
   3) 函数 int freeBlock(char* buf)：
      - 内存池未初始化成功，无法释放；
      - buf 参数空或者错误；
      - 如果内存池需要扩展，使用 realloc 函数会导致无法回收内存块。
   4) 函数 freePool()以及函数 freePoolForce()：：
      - 内存池未初始化成功，无法释放；
      - 同样存在释放完内存不将指针置为空，导致也野指针的问题。
   5) 函数 getBlockCount()以及函数 getAvailableBlockCount()：
      - 内存池未初始化成功，内存头指针为空，无法获得数据；

2. 设计思路：

   我设计了一个带头结点（mypool）的内存池链，其节点为结构体 Mem_Block（block），结构体包含以下变量：

   int No;//编号，对于头节点，存放总节点（总内存块数）个数
   char *pm;//指向内存块（256 个字节）
   Mem_Block* next;//指向下一个节点
   bool free ;//true 表示空闲 ， false 表示已分配



1) int initPool(size_t size)

流程：

顺序	步骤
①	内存池头节点初始化：`mypool = (block*)malloc(sizeof(block));` 申请成功继续下一步，否则返回-1； 头节点的 mypool->No 存放内存块数 size，mypool->next 为空。
②	For 循环进行逐块申请内存： 对于每个 block 节点初始化申请 sizeof(block)的内存空间； 每个 block 节点的内存块指针初始化申请 256 个字节的空间； 如果某一步申请失败则回收前面申请的全部内存（包括头节点），并将指针置空，返回-1； 如果申请成功，执行下一步；
③	每申请一块就插入头节点之后，节点的 No 存入编号 size+1-i（由于是插入头节点后，故是倒序，编号不为 i），节点的 free 置为 true 表示未分配。返回 0。

2) char* allocBlock()

使用插入新节点来扩充内存，以避免 realloc 带来的风险。

流程：

顺序	步骤
①	如果头节点为空，返回空；
②	while 循环判断内存池空是否有空闲块，若某一块的 free 为 true，则对该块进行分配： free 置为 false， 使用 memset(f->pm, 0, 256)置为 0， 返回内存块首址 pm；
③	循环内同时设置一个 block 指针 fpre 循环结束指向最后一块； 扩充新节点失败则返回 NULL，若扩充新块失败同样返回 NULL，同时回收新节点申请的空间。
④	若成功则标号为 mypool->No,且头节点 No 加一，新节点 free 置为 false；用 memset 函数置为 0，next 置空。
⑤	新块插入在 fpre 后（插在最后是为了使整个链有序），返回分配的内存指针。

3) int freeBlock(char* buf)

流程：

顺序	步骤
①	如果头节点或 buf 参数为空，返回-1；
②	while 循环找到 buf 所指的块，并且其 free 值为 false， memset 清零，free 置为 true，并返回 0；否则返回-2。

4) int freePool()

流程：

顺序	步骤
①	如果头节点为空，返回-1；
②	while 循环找是否存在未回收的块，若存在，返回-1；
③	如果均已回收，则 while 循环清零、释放内存块和 block 节点，分别置空；
④	释放头节点，置空，返回 0。

5) int freePoolForce()
流程：

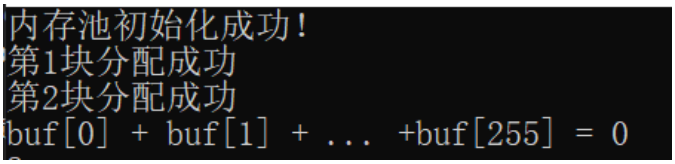
顺序	步骤
①	如果头节点为空，返回-1；
②	while 循环找是否存在未回收的块，若存在，调用 freeBlock 函数回收；
③	while 循环清零、释放内存块和 block 节点，分别置空；
④	释放头节点，置空，返回 0。

6) size_t getBlockCount()
流程：
如果头节点为空，返回 0；否则返回 mypool->No。

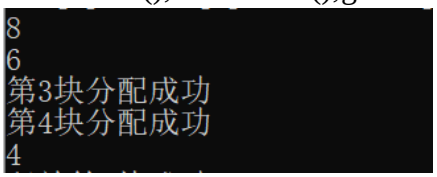
7) size_t getBlockCount()
流程：
如果头节点为空，返回 0；否则 while 循环计算可分配节点。

8) 测试用例及结果：

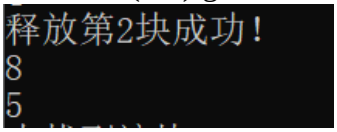
- initPool(8);allocBlock();char* buf = allocBlock();



- getBlockCount();getAvailableBlockCount();
allocBlock();allocBlock();getAvailableBlockCount();



- freeBlock(buf);getBlockCount();getAvailableBlockCount();



- freeBlock(buf);

```
未找到该块!
```

- freePool()==-1;freePoolForce();
  getBlockCount()==0;getAvailableBlockCount()==0;

```
释放内存池失败，有未回收的内存块。
freePool()==-1
释放第1块成功!
释放第3块成功!
释放第4块成功!
freePoolForce成功!
freePoolForce()==0
0
0
```

3. 心得体会

本次实验，了解了内存池这一概念，内存池的产生使得内存碎片的问题得以解决并且相对效率高，个人感觉其思想——先提取再使用在各个地方都有用到，通过一些课外阅读，我了解到其他的诸如连接池、线程池，这对于我以后学习其他知识举一反三也是有极大的帮助；其次也体会到内存分配释放的操作十分繁琐，需要小心再小心。内存池使用链表的方式，也加深了我对数据结构链表应用的理解。