计算机系统体系结构 Project7

517030910326 王孝诚

2019.11.30

实验环境

Windows 10 下使用 VMWare Workstation 15 Player 创建和运行虚拟机，虚拟机环境是 Linux 发行版 Ubuntu16.04.6 LTS。

1 连续内存分配 Contiguous Memory Allocation

将要实现一个连续内存分配程序，给定内存大小，完成其四个功能：

1. 请求分配指定大小内存
2. 释放指定进程的内存
3. 将未使用的内存压缩到一块
4. 报告内存分配情况

1.1 设计思路

1. **分配内存：** 在分配内存时使用指定的分配算法：first fit, best fit 或者 worst fit。要追踪每个内存块占用的大小，以及占用者，所以考虑维护两个数组 memoryHoles 和 allocatedName，前者的元素为每个块的大小，后者的元素为每个对应块的占用进程的名字。当分配块内存后，有可能产生新的内存块，则要在数组中部插入新元素，这里的解决方法是将插入位置以后的元素整体移动。
2. **释放内存：** 释放内存可以直接通过指定的进程序号找到相应块，改变其占用者为空即完成了释放。同时，若其相邻的内存块为空，这里需要直接对他们进行合并。合并后，维护的两个的数组的元素也要进行相应移动。
3. **压缩内存：** 压缩内存的操作主要也是在对维护的两个数组进行移动和合并的操作。首先遍历数组，将被占用的内存块移到左边，相应的空内存块就堆积在了右边。然后将空内存块的容量加起来，形成一个大的空内存块。
4. **打印内存状态：** 将 memoryHoles 和 allocatedName 的元素整合输出即可。

1.2 核心代码解释

1. 维护的全局变量

```
#define EMPTY "Unused"
int memorySize;
int memoryHoles[100];
char *allocatedName[100];
int holeNum;
```

2. 申请内存

```
int allocateMemory(char *name, int size, char* flag)
{
    if (flag[0] == 'F'){ // first fit
        for (int i=0; i<holeNum; i++){
            if (!strcmp(allocatedName[i], EMPTY)){
                if (size < memoryHoles[i]){ // 产生新hole
                    for (int j=holeNum; j>i; j--){ //
                        左边产生新hole，整体右移一位
                        allocatedName[j] = allocatedName[j-1];
                        memoryHoles[j] = memoryHoles[j-1];
                    }
                    memoryHoles[i+1] = memoryHoles[i] - size;
                    memoryHoles[i] = size;

                    allocatedName[i] = malloc(sizeof(char)*strlen(name));
                    strcpy(allocatedName[i], name);
                    holeNum ++;
                    return 1;
                }
                else if(size == memoryHoles[i])
                {
                    allocatedName[i] = malloc(sizeof(char)*strlen(name));
                    strcpy(allocatedName[i], name);
                    return 1;
                }
            }
        }
        return 0;
    }
    if(flag[0] == 'W'){
        int max_pos = -1;
        int crt_max = size-1;
        for(int i=0; i<holeNum; i++){
            if (!strcmp(allocatedName[i], EMPTY)){
                if (memoryHoles[i]>crt_max){
                    max_pos = i;
                    crt_max = memoryHoles[i];
                }
            }
        }
        if(max_pos == -1){
            return 0;
        }
        if(size < crt_max){
            for (int j=holeNum; j>max_pos; j--){ // 右移
                allocatedName[j] = allocatedName[j-1];
                memoryHoles[j] = memoryHoles[j-1];
            }
            memoryHoles[max_pos+1] = memoryHoles[max_pos] - size; // 新hole
            memoryHoles[max_pos] = size;
```

```
50
51              allocatedName[max_pos] = malloc(sizeof(char)*strlen(name));
52              strcpy(allocatedName[max_pos], name);
53              holeNum++;
54          }
55          else if (size == crt_max){
56              allocatedName[max_pos] = malloc(sizeof(char)*strlen(name));
57              strcpy(allocatedName[max_pos], name);
58          }
59      }
60      if (flag[0] == 'B'){
61          int min_pos = -1;
62          int crt_min = memorySize+1;
63          for (int i=0; i<holeNum; i++){
64              if (!strcmp(allocatedName[i], EMPTY)){
65                  if (memoryHoles[i]>=size){
66                      if(memoryHoles[i]<crt_min){
67                          min_pos = i;
68                          crt_min = memoryHoles[i];
69                      }
70                  }
71              }
72          }
73          if (min_pos == -1){
74              return 0;
75          }
76          if (size < crt_min){
77              for (int j=holeNum; j>min_pos; j--){
78                  allocatedName[j] = allocatedName[j-1];
79                  memoryHoles[j] = memoryHoles[j-1];
80              }
81              memoryHoles[min_pos+1] = memoryHoles[min_pos] - size;
82              memoryHoles[min_pos] = size;
83
84              allocatedName[min_pos] = malloc(sizeof(char)*strlen(name));
85              strcpy(allocatedName[min_pos], name);
86              holeNum++;
87          }
88          else if (size == crt_min){
89              allocatedName[min_pos] = malloc(sizeof(char)*strlen(name));
90              strcpy(allocatedName[min_pos], name);
91          }
92      }
93  }
```

3. 释放内存

```
1   int releaseMemory(char *name)
2   {
3       for (int i=0; i<holeNum; i++){
4           if (allocatedName[i][1]==name[1]){
5               allocatedName[i] = EMPTY;
```

```
            int pos = i;
            if (pos>0 && !strcmp(allocatedName[pos-1], EMPTY)){ // 与左边的空hole合并
                memoryHoles[pos-1] += memoryHoles[pos];
                for (int j=pos; j<holeNum-1; j++){
                    memoryHoles[j] = memoryHoles[j+1];
                    allocatedName[j] = allocatedName[j+1];
                }
                holeNum--;
                pos--;
            }
            if (pos<holeNum-1 && !strcmp(allocatedName[pos+1], EMPTY)){ // 与右边的空hole合并
                memoryHoles[pos] += memoryHoles[pos+1];
                for (int j=pos+1; j<holeNum-1; j++){
                    memoryHoles[j] = memoryHoles[j+1];
                    allocatedName[j] = allocatedName[j+1];
                }
                holeNum--;
            }
            return 1;
        }
    }
    return 0;
}
```

4. 压缩内存

```
void compactMemory()
{
    int count = 0;
    for (int i=0; i<holeNum; i++){
        if (strcmp(allocatedName[i], EMPTY)){
            int pos = i;
            while(pos>0 && !strcmp(allocatedName[pos-1], EMPTY)){ // 将该进程移到左边
                allocatedName[pos-1] = allocatedName[pos];
                int tmp = memoryHoles[pos-1];
                memoryHoles[pos-1] = memoryHoles[pos];
                allocatedName[pos] = EMPTY;
                memoryHoles[pos] = tmp;
                pos--;
            }
        }
    }
    for (int i=0; i<holeNum; i++){
        if (!strcmp(allocatedName[i], EMPTY)){ // 将所有空hole合并
            int pos = i;
            int tmp = 0;
            while (pos < holeNum){
                tmp += memoryHoles[pos++];
            }
```

```
24                memoryHoles[i] = tmp;
25                holeNum = i+1;
26                return;
27            }
28        }
29    }
```

1.3 实验结果

```
allocator>STAT
Addresses [0:999] Process P0
Addresses [1000:999999] Unused
allocator>RQ P1 1000 W
allocator>RQ P2 1000 W
allocator>STAT
Addresses [0:999] Process P0
Addresses [1000:1999] Process P1
Addresses [2000:2999] Process P2
Addresses [3000:999999] Unused
allocator>RL P1
allocator>S
Addresses [0:999] Process P0
Addresses [1000:1999] Unused
Addresses [2000:2999] Process P2
Addresses [3000:999999] Unused
allocator>C
allocator>S
Addresses [0:999] Process P0
Addresses [1000:1999] Process P2
Addresses [2000:999999] Unused
```

Figure 1: 示例