武汉大学计算机学院

课程实验(设计)报告

专业(班)：计算机科学与技术 三班

学　　号：2013301500100

姓　　名：秦贤康

课程名称：操作系统设计

任课教师：宋伟

2015 年　　11 月　　26 日

说　　明

1、本课程设计报告包括：实习题目、实习内容及设计思想（设计思路、主要数据结构、主要代码结构及代码段分析）、上机实习所用平台及相关软件、调试过程（测试数据设计、测试结果分析）、总结（实习中遇到的问题及解决方法、实习中产生的错误及原因分析、实习体会及收获）。

2、课程设计报告用 A4 纸打印。

标题 4 号宋体加粗，正文 5 号宋体，图表文字标注、程序等用 6 号。

页边距：左边 3.17cm，右边 3.17 cm，上 2.54 cm，下 2.54 cm。

行间距：1.5 倍行距，一页 30 行，每行 39 字。

页脚：1.5 cm，页脚(页码，居中)。

主存空间的分配和回收

1 实习内容

主存储器空间的分配和回收。

2 实习题目及设计思想

2.1 实习题目

第一题：可变分区管理方式下采用首次适应算法实现主存分配和回收

[提示]:

可变分区方式是按作业需要的主存空间大小来分割分区的。当要装入一个作业时，根据作业需要的主存容量查看是否有足够的空闲空间，若有，则按需分配，否则，作业无法装入。假定内存大小为128K(可输入)，空闲区说明表格式为：

·起始地址——指出空闲区的起始地址；

·长度——一个连续空闲区的长度；

·状态——有两种状态，一种是“未分配”状态；另一种是“空表目”状态，表示该表项目前没有使用。

采用首次适应算法分配回收内存空间。运行时，输入一系列分配请求和回收请求。

要求能接受来自键盘的空间申请及释放请求，能显示分区分配及回收后的内存布局情况。

2.2 设计思路

·把类的对象指针队列保存到类里面

·每次某个进程运行之后，按队列中内存块的开始位置

·每次申请分配队列中第一个符合条件的空内存块

·作业 id 递增,保持 id 唯一

2.3 主要数据结构

```
class Task{
private:int start;                                  //起始位置
        int task_size;                              //作业内存大小
        int end;                                    //结束位置
        int id;                                     //作业 id
        bool state;                                 //作业状态
public:
        static int size;                            //内存大小
        static list<Task*> task_list;               //作业列表
        static int taskNum;                         //分配作业 id,保证 id 唯一
        Task(int st, int ed);                       //用于测试的方法
        Task(int t_size);                           //构造函数
```

```
    static void showMem();                                  //显示内存分配状况
    bool call();                                            //申请作业内存
    static bool callback(int id);                           //释放作业
    void disp();                                            //显示当前作业信息
    static bool compare_task(const Task* f, const Task* s); //内存块排序
    static void setSize(int s);                             //设置内存大小
};
```

2.4 主要代码结构及代码段分析

```
/**
* @brief 作业申请内存
*/
bool Task::call()
{   int temp_size=0,temp_pos=0,pos=0;
    list <Task*>::iterator plist;
    if(task_list.empty())                                   //如果作业为空、
    {   /*暂留,为以后的情况考虑*/
    }
    for(plist = task_list.begin(); plist != task_list.end(); plist++)
    {   Task * task;
        task = *plist;
        pos = task->start;
        #ifdef DEBUG
        task->disp();
        cout<<pos<<temp_pos<<task_size<<endl;
        #endif
        if(pos - temp_pos >= task_size)                     //如果找到第一个合适的空间大小
        {   this->state = true;
            this->start = temp_pos;
            this->end = temp_pos+this->task_size;
            task_list.push_back(this);                      //把作业存入列表
            task_list.sort(compare_task);                   //按地址顺序进行排序
            return true;
        }
        temp_pos = task->end;
    }
    if(Task::size - temp_pos >= task_size)                  //处理最后一个空作业
    {   this->state = true;
        this->start = temp_pos;
        this->end = temp_pos+this->task_size;
        task_list.push_back(this);
        task_list.sort(compare_task);
        return true;
    }
    #ifdef INFO
```

```
        cout<<"内存申请失败:找不到大小为"<<task_size<<"的空闲块"<<endl;
        #endif
        return false;
}
```

申请内存块的时候，需要遍历作业列表，申请成功之后，加入作业队列
空间不够时申请失败。

```
/**
* @brief 释放作业
* @param id 作业 id
* @return 是否释放成功
*/
bool Task::callback(int id)
{       list <Task*>::iterator plist;
        if(task_list.empty())
        {
                #ifdef INFO
                cout<<"内存空间为空"<<endl;
                #endif
                return false;
        }
        for(plist = task_list.begin(); plist != task_list.end(); plist++)
        {       Task * task;
                task = *plist;
                if(id == task->id)
                {       task_list.remove(task);
                        return true;
                }
        }
        #ifdef INFO
        cout<<"内存释放失败:找到不进程"<<id<<endl;
        #endif
        return false;
}
```

释放作业时，需要先判断作业是否存在。
释放作业之后，应该考虑合并空块，但是我用的是另外一种方法，也就没有合并这一步。

3 上机实习所用平台及相关软件

OS: mint Mint 17.2 rafaela
Kernel: x86_64 Linux 3.16.0-38-generic
CPU: AMD A8-4500M APU with Radeon HD Graphics @ 1.9GHz
GPU: Gallium 0.4 on AMD ARUBA
RAM: 1522MiB / 3337MiB
Vim:用 doxygen 自动生成注释
g++ 4.8.4

4 调试过程

Test.sh

```
#!/bin/sh
# 测试三条数据
cat test1|./mem >test1.out
```

运行 test.sh 自动进行输入输出，只要事先把测试数据写到 test 文件中就可以了

4.1 测试数据设计

Test1

```
1024
1 24
1 46
1 500
1 600
2 2
1 30
1 500
2 9
0
```

初始化 1024 大小的内存块
1 申请，后接大小
2 释放作业内存，后接作业 id
0 退出

4.2 测试结果分析

内容有点多，只以部分截图为例说明，详细结果可以看 test1 输入对应的结果 test1.out

请输入内存块大小

```
   0|----------
    | Empty
1024|----------
```

说明：初始化内存大小为 1024
略过数步申请，之后结果如下：

```
   0|----------
    | Task:0
  24|----------
    | Task:1
  70|----------
    | Task:2
 570|----------
    | Empty
1024|----------
```

```
--------------------------------
      1:申请作业内存
```

2:回收作业

0:退出

请输入:

此时申请作业大小为 600 的空闲块，应该会失败

输入作业大小:内存申请失败:找不到大小为 600 的空闲块

```
   0|----------
    | Task:0
  24|----------
    | Task:1
  70|----------
    | Task:2
 570|----------
    | Empty
1024|----------
```

1:申请作业内存

2:回收作业

0:退出

请输入:

输入回收作业 ID:

```
   0|----------
    | Task:0
  24|----------
    | Task:1
  70|----------
    | Empty
1024|----------
```

1:申请作业内存

2:回收作业

0:退出

请输入:

输入作业大小:

```
   0|----------
    | Task:0
  24|----------
    | Task:1
  70|----------
    | Task:4
 100|----------
    | Empty
```

1024|----------

1:申请作业内存

2:回收作业

0:退出

请输入:

输入作业大小:

0|----------

| Task:0

24|----------

| Task:1

70|----------

| Task:4

100|----------

| Task:5

600|----------

| Empty

1024|----------

1:申请作业内存

2:回收作业

0:退出

请输入:

目前只有０１４５几个作业，申请释放作业９会失败

输入回收作业 ID:内存释放失败:找到不进程 9

0|----------

| Task:0

24|----------

| Task:1

70|----------

| Task:4

100|----------

| Task:5

600|----------

| Empty

1024|----------

1:申请作业内存

2:回收作业

0:退出

请输入：**退出**

实时运行截图结果如下：

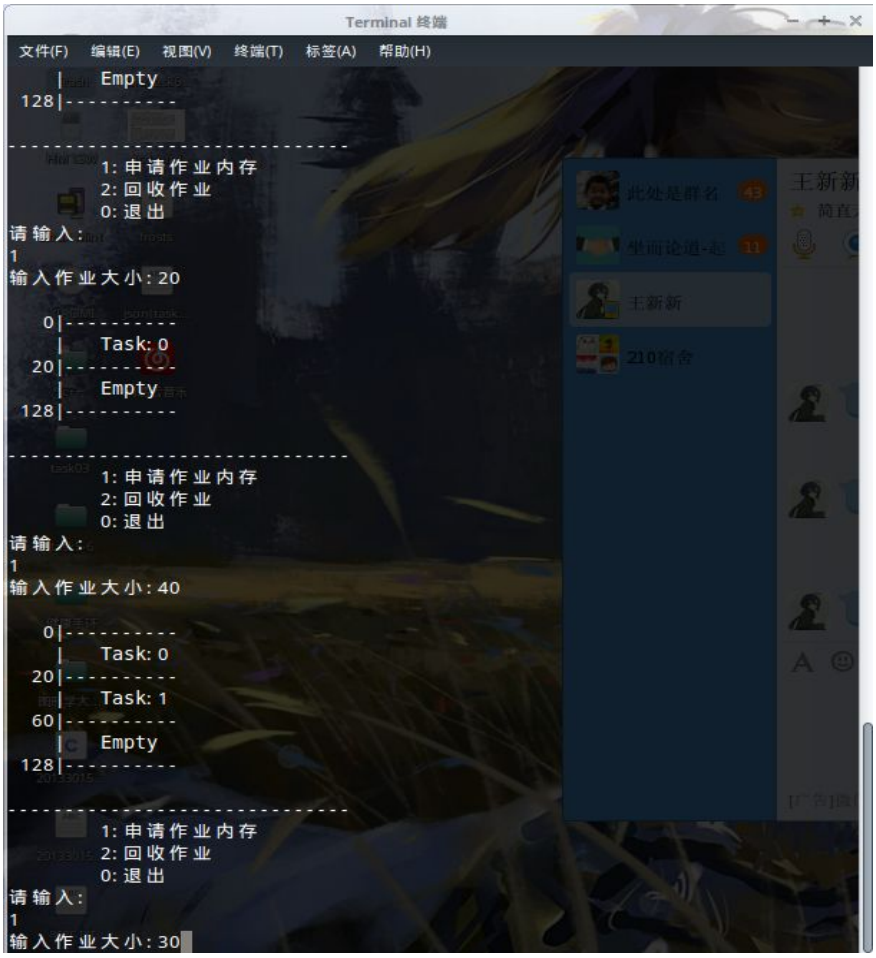

图 4-1

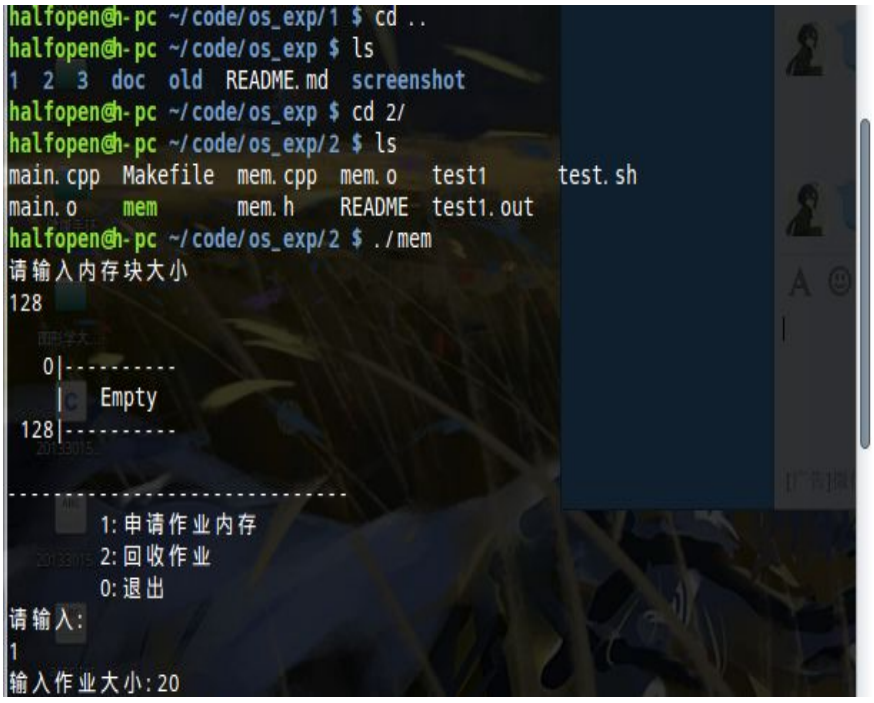

图 4-2


Terminal 终端
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
2: 回收作业
0: 退出
请输入:
1
输入作业大小: 30
0|----------
| Task: 0
20|----------
| Task: 1
60|----------
| Task: 2
90|----------
| Empty
128|----------
1: 申请作业内存
2: 回收作业
0: 退出
请输入:
2
输入回收作业ID: 0
0|----------
| Empty
20|----------
| Task: 1
60|----------
| Task: 2
90|----------
| Empty
128|----------
1: 申请作业内存
2: 回收作业
0: 退出
请输入:

图 4-3

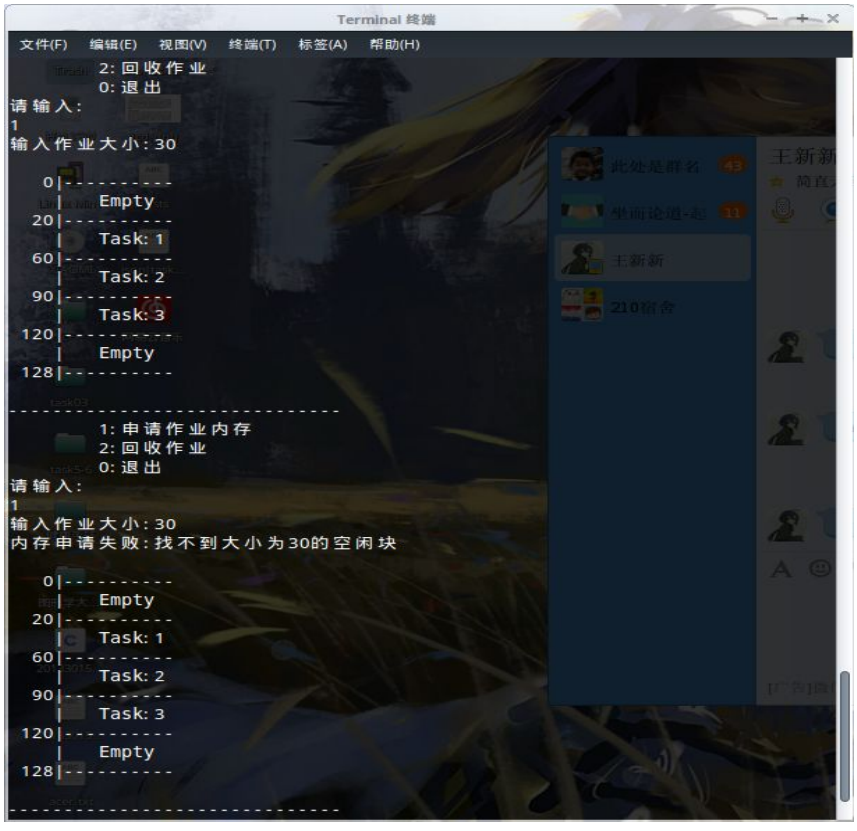
Terminal 终端
文件(F) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
2: 回收作业
0: 退出
请输入:
1
输入作业大小: 30
0|----------
| Empty
20|----------
| Task: 1
60|----------
| Task: 2
90|----------
| Task: 3
120|----------
| Empty
128|----------
1: 申请作业内存
2: 回收作业
0: 退出
请输入:
1
输入作业大小: 30
内存申请失败: 找不到大小为30的空闲块
0|----------
| Empty
20|----------
| Task: 1
60|----------
| Task: 2
90|----------
| Task: 3
120|----------
| Empty
128|----------

图 4-4

图 4-5

由上结果可以看出，此次实验顺利模拟了可变分区管理方式下采用首次适应算法实现主存分配和回收

5 总结

本次实验帮助了我理解在不同的存储管理方式下应怎样进行存储空间的分配和回收。我看到有些代码把空闲块合并写了很多判断，只要释放完了之后，把上一个已经分配块的结束位置，下一直已分配块的开始位置，这两个位置作为新空闲块的起始位置就行了。

遇到的问题及解决方法:

1、主存分区的显示问题

因为情况比较多，写得比较麻烦，自己把逻辑理清楚就好了

2、格式问题

把代码传到 github 上去之后，网页上的 README.md 一直显示不了换行符

解决方法：

A):iconv 格式转换成 gbk windows 格式的，没有效果

B)Vim set fileencoding=utf8,没有效果

最后，把 README 的后缀名去掉，网页就不会把它当作 markdown 文件处理了，换行问题解决

总结:备份代码很重要，写代码的时候也许可以先不写注释，但写完之后，一定要有一步就是添加注释，可以用 vim 的 doxygen 插件自动生成。在写判断的时候，要尽量多地考虑各种情况，某种情况就算没有想到也可以先写上。

个人觉得写测试语句可以用上预编译提高效率。

A)