东北大学秦皇岛分校

计算机与通信工程学院

操作系统课程设计

专业名称	计算机科学与技术
班级学号	20188068
学生姓名	**孔天欣**
指导教师	王军伟
设计时间	**2020 年 12 月 13 日—2020 年 12 月 23 日**

课程设计任务书

专业：计算机科学与技术　学号：20188068　学生姓名（签名）：

设计题目：操作系统课程设计

一、设计实验条件

基于 Windows10 的 Qt 和 IDE

二、设计任务及要求

本次课程设计的任务是虚拟实现部分操作系统的典型算法，加深对操作系统运行机制的掌握和理解，要求实现置换策略、目录管理、外存管理和空闲磁盘管理。本组任务具体要求如下表。

置换策略	目录结构	外存组织	空闲磁盘管理
全局置换 LRU	二级目录	一级索引	位示图

三、设计报告的内容

1. **设计题目与设计任务（设计任务书）**
2. **前言（绪论）(设计的目的、意义等)**
3. **设计主体（各部分设计内容、分析、结论等）**
4. **结束语（设计的收获、体会等）**
5. **参考资料**

四、设计时间与安排

1、设计时间：　2 周

2、设计时间安排：

熟悉实验设备、收集资料：　2　天

设计图纸、实验、计算、程序编写调试：　9　天

编写课程设计报告：　2　天

答辩：　1　天

前 言

操作系统（Operating System）是一组主管并控制计算机操作、运用和运行硬件、软件资源和提供公共服务来组织用户交互的相互关联的系统软件程序，同时也是计算机系统的内核与基石。操作系统需要处理如管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务。操作系统也提供一个让用户与系统交互的操作界面。

自主设计操作系统，非常考验个人操作系统能力的基本功。通过在不断的调试和改错中实现操作系统的各个基本功能，能使得个人够对操作系统本身的机制有更加深入的理解。通过课本上的理论和实际操作系统实践相互结合，能够使个人提高自身的代码水平，并理解操作系统各项机制为何如此设计的原因。

本次课程设计主要是模拟操作系统的各个功能，并将它们组装连接到一起，从而整体实现操作系统的基本操作，最后还实现了人机交互的简易可视化功能。本人主要负责操作系统各项功能中的一级索引模块。通过模拟外存和盘块的实现，实现一级索引。一级磁盘索引结构中，在盘块组中有专门的盘块用于存放索引，每一个索引登记的是文件记录所在的磁盘块号，一级间接索引结构中，还新增了指向记录更多该文件索引盘块号的索引指针，可以实现文件上限容量的扩大。

目 录

操作系统课程设计报告

1 设计主体

1.1 设计内容

本次课程设计的任务是虚拟实现部分操作系统的典型算法，加深对操作系统运行机制的掌握和理解。

在操作系统整体功能中，本文实现了磁盘管理以及一级索引的模块功能。磁盘管理以及一级索引要求如下：

建立一个 4MB 大小的文件模拟磁盘，按逻辑将其划分为 1024 块，每块大小 4KB。其中 900 块用于存放普通数据，124 块用于存储对换数据。存储管理支持：

（1）数据组织：对需要存放的文件数据加以一级索引管理方式。

（2）空闲块管理：能够查询并返回当前剩余的空闲块，对空闲块管理采用位示图法（实践上为提高空间利用率，且能实现相同目的，将位示图和硬盘块数据结构予以合并）。

（3）兑换区管理：能够写入、读出对换区数据。

1.2 设计分析

如果对文件大小上限容量要求不大，那么可以直接使用一个索引块存储单个文件各个记录所在盘块的索引指针集，如图 1.2.1 所示。

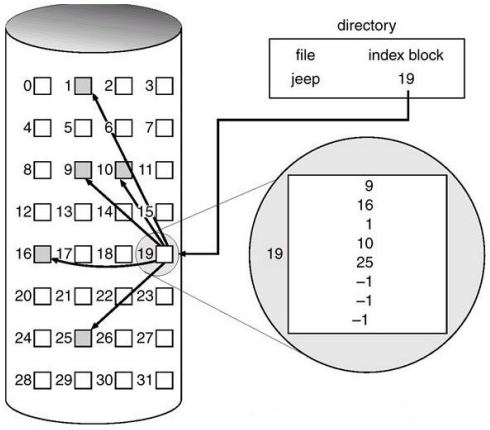

图 1.2.1 单级索引组织方式

如果需要扩大文件的容量上限，那么需要使用多级索引存储方式。在该模拟操作系统中使用了一级磁盘索引结构。一级间接索引结构中，还新增了指向记录更多该文件索引盘块号的索引指针，可以实现文件上限容量的扩大，如图 1.2.2 所示。

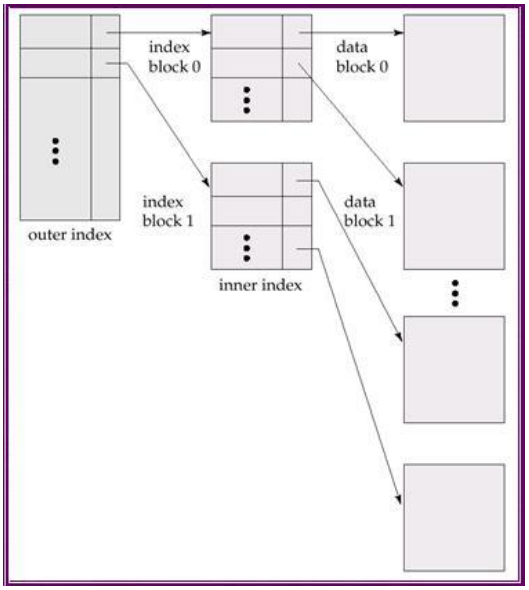

图 1.2.2 一级间接索引组织方式

本文从源代码入手，分析外存磁盘管理模块的运行过程。

该操作系统使用 disk_manager.h 和 disk_manager.cpp 作为磁盘管理系统的核心代码，并使用 disk_monitor.h 和 disk_monitor.cpp 进行磁盘内容可视化的呈现。下述代码内容主要基于 disk_manager.cpp 进行分析。

1.2.1 索引块 Index_File

一级索引就是使用盘块用于存放文件的各个记录索引，当读取文件的记录时，先读出该索引盘块，然后根据该盘块内存放的的各个索引指针去寻找对应存放文件数据的盘块，并将数据取出装入内存。外存索引块 Index_File 的数据结构如下所示。

```
1.  typedef struct Index_File
2.  {
3.
4.      string fileName;
5.      int fileSize;
6.      int addr[10] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
7.      Index_block_three *addr10;
8.
9.      // 序列化函数
10.     template<class Archive>
```

```
    void serialize(Archive & ar){
        ar(CEREAL_NVP(fileName),CEREAL_NVP(fileSize),CEREAL_NVP(addr),CEREAL_NVP(addr10));
    }

}Index_File;
```

在 Index_File 内，filename 是块名，filesize 是块大小，addr 是一个长度为 10 的数组，每一个元素都是一个盘块号的直接指针索引，读取文件时，操作系统读取文件的索引盘块，然后通过 addr 中存储的每个盘块号去寻找对应的记录盘块，取出其中的记录。此外，规定在 addr 数组中，元素为 -1 的情况下代表尚未初始化，因此并没有任何索引记录。

1.2.2 一级索引块 Index_Block_One

如果分配出去的盘块号数量较多，那么需要为索引块再次建立一级索引，因此设计了 Index_block_one，这是一个一级索引块的数据结构，该数据结构所产生的对象就是模拟外存中的一级索引块，在 Index_File 中，存在一个指向该一级索引块的指针 addr10，如果盘块号的数量超过了 10，那么就需要使用该一级索引块，将多余的盘块号记录索引指针存储到其中，这样就可以实现扩大单文件容量上限的效果。对于 Index_block_one，它的数据结构如下所示。

```
struct Index_block_one
{
    int blocks[MAX_NUMBER_IN_BLOCK];

    template<class Archive>
    void serialize(Archive & ar){
        ar(CEREAL_NVP(blocks));
    }
};
```

可以明显看出，该数据结构中的整型数组 blocks 就是用于存储文件记录的盘块号，其中的每一个元素都指向一个盘块号。

1.2.3 磁盘管理系统 DiskManager

该操作系统要求实现建立一个 4MB 大小的文件模拟磁盘，按逻辑将其划分为 1024 块，每块大小 4KB。其中 900 块用于存放普通数据，124 块用于存储兑换数据，因此对应的常量定义如下。

```
#define BlockNum     1024
#define BlockSize    4
#define SystemSize   1024*4
```

其中 BlockNums 是外存中盘块的数目 1024，BlockSize 是每块大小 4KB，因此整个外存的容量 SystemSize 是 1024 * 4 的大小。

对于该磁盘管理系统，需要有恢复数据和读取数据的能力，因此在磁盘管理系统进行初始化的时候，需要将模拟外存中的数据从 json 文件中进行读取和反序列化并装入内存，同时，在退出磁盘管理系统或者磁盘管理系统执行的时候，需要将模拟外存中的数据进行序列化，以方便数据的持久化存储。

磁盘管理系统的初始化构造函数部分如下。

```
DiskManager::DiskManager()
{
    std::ifstream os("disk.json");
    if (os.is_open()){
        cereal::JSONInputArchive archieve(os);
        this->serialize(archieve);
    } else {
        for(int i=0;i<1024;i++){
            Map[i].isFree = true;
            Map[i].x = (i)/32;
            Map[i].y = (i)%32;
            Map[i].data = nullData;
            disk[i] = nullData;
        }
    }
}
```

可以看到，在初始化的时候，首先判断持久化文件 disk.json 是否存在，如果存在，那么就打开文件输入流对 disk.json 文件进行读取，里面的数据经过反序列化之后成为了模拟外存中的盘块数据。如果不存在，说明磁盘系统是最新启动的，那么就初始化位示图数组 Map，主要靠 isFree 进行推断盘块号是否被占用。Map 对象的 x 和 y 主要用于可视化坐标定位，data 用于存储记录数据，在该模拟操作系统中，data 最多可存 4 个字符的数据，用来模拟 4KB 的盘块大小。同时，为保持磁盘系统的独立性，另外设置磁盘块数组 disk 用来存放数据。

1.2.4 磁盘分配管理

在用户创建一个文件的时候，会预输入一些数据来确定整个文件的大小，因此确定对应分配的盘块号数量，相关代码如下所示。

```
Index_File DiskManager::indexFile(int filesize)
{
    int block_num = filesize % BLOCK_SIZE == 0 ? filesize / BLOCK_SIZE : filesize / BLOCK_SIZE + 1;
    int blocks[block_num];
    memset(blocks,0,sizeof(int) * block_num);

    for(int i = 0; i < block_num; i++)
    {
        int temp = rand() % MAX_BLOCK_NUMBER;
        while(this->Map[temp].isFree == false)
        {
            temp = rand() % MAX_BLOCK_NUMBER;
        }
        this->Map[temp].isFree == false
        blocks[i] = temp;
    }

    Index_File indexfile;
    indexfile.fileSize = filesize;
    if(block_num <= 10)
    {
        for(int i = 0; i < block_num; i++)
        {
            indexfile.addr[i] = blocks[i];
        }
    }
    else if(block_num <= MAX_NUMBER_IN_BLOCK + 10)
    {
        for(int i = 0; i < 10; i++)
        {
            indexfile.addr[i] = blocks[i];
        }
        indexfile.addr10 = indexBlockOne(blocks,10,block_num);
    }

    return indexfile;
}
```

该函数就执行上述的根据文件大小分配盘块号行为，首先是确定 blockNum，将文件的大小除以一个盘块的大小，就可以获得盘块的数量，然后初始化盘块数组，进行盘块的分配，如果盘块的数量小于等于 10，就直接分配，否则需要使用到一级索引块对象。

盘块分配的过程可以如此简述：首先，遍历每个盘块数组的元素，对于每一个元素，都使用一个随机函数，范围是 0~ MAX_BLOCK_NUMBER（即 1024），随机获得一个盘块号，如果该盘块号对应的位示图中该块已经被占用，那么就一直随机，直到随机出一个空闲盘块号为止。使用反复随机的原因主要是顺序访问容易造成占用盘块在低地址区间的集中，长久以来再次分配导致的遍历访问会造成时间效率的下降，而随机法虽然时间确定性无法保证，但空闲盘块的数量越多，随机到空闲盘块的几率越大，因此可以使用。寻找到未被占用的盘块号后，就把它赋给盘块号数组。直到分配完所有盘块号后遍历结束。

之后，就创建该文件专属的索引块，里面存放分配完毕的所有盘块号，如果盘块号的数量大于 10，那么就将前 10 个索引先直接放入该索引块，然后再创建一个一级索引块，将剩下的盘块号放入其中，在代码清单中，就是执行 indexBlockOne 方法，该方法如下所示。

```
1.  Index_block_one indexBlockOne(int blocks[],int start,int end)
2.  {
3.      Index_block_one ans;
4.      int j = 0;
5.      for(int i = start; i < end; i++)
6.      {
7.          ans.blocks[j++] = blocks[i];
8.      }
9.      return ans;
10. }
```

在该方法中，首先创建了一个一级索引块对象。形参中 start 固定是 10，也就是从第 10 个盘块号开始赋给一级索引块的 blocks 数组，end 就是盘块号的总量。

1.2.5 对换区磁盘管理

该模拟操作系统要求 900~1024 盘块为对换区，因此操作系统要求换入后，磁盘管理模块就执行换入策略。相关代码如下。

```
int DiskManager::receiveM(FCB* e,int pageNumber,string data){
    if (pageNumber == -1){
        return STATUS_OK;
    }
    for(int i=900;i<1024;i++){
        if(pageNumber == this->Map[i].pageNumber && e->fileName == this->Map[i].fileName){
            return STATUS_OK;
        }
    }
    changeBlock(e,data,pageNumber);
    return STATUS_OK;
}
```

进行换入时，需要接收到 FCB 控制块对象，以及数据和页号，并将相关数据放入内存。首先判断页号，如果页号为空，那么就直接返回。接着即遍历对换区的所有盘块位示图，如果存在对换区盘块位示图中的页号和文件名和形参中传入的一样，说明已经被换入，返回对换成功 STATUS_OK 标志，那么就如果存在空闲盘块，就把数据装入到空闲盘块当中去。如果找不到空闲盘块，就返回 STATUS_BUSY 标志，表示对换区分配失败。相关代码如下。

```
int DiskManager::changeBlock(FCB *e,string data,int number){
    for(int i=900;i<1024;i++){
        if(this->Map[i].isFree){
            this->Map[i].isFree = false;
            this->Map[i].fileName = e->fileName;
            this->Map[i].data = data;
            this->Map[i].pageNumber = number;
            this->disk[i] = data;
            return STATUS_OK;
        }
    }
    return STATUS_BUSY;
}
```

进行换出时，相关代码也是类似的，只要遍历对换区中的位示图，并询问每个元素中文件名和页号的情况和形参中的传入的页号是否相符，如果有对应相符合的，那么就将其换出，然后将该盘块置为空闲。

```
string DiskManager::returnM(FCB* e,int number){
    string dataReturn;
    for(int i=900;i<1024;i++){
        if(number == this->Map[i].pageNumber && e->fileName == this->Map[i].fileName){
            dataReturn = this->Map[i].data;
            ReleaseBlock(e);
            return dataReturn;
        }
    }
    return dataReturn;
}
```

1.2.6 目录-磁盘管理

当目录系统申请空间时，磁盘管理系统应当将相关的盘块分配给它，然后统一对位示图进行修改，相关代码如下。

```
int DiskManager::receiveF_add(FCB *e,string data){
    // data 在 UI 上限定死了，最大为 48
    int a;
    a = data.size();
    e->iFile = indexFile(a);

    int fnumber = e->fileSize;
    int b[12] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
    for(int i=0;i<10;i++){
        b[i] = e->iFile.addr[i];
    }
    if(fnumber==11)
    {
        b[10] = e->iFile.addr10.blocks[0];
    }else if(fnumber>11)
    {
        b[10] = e->iFile.addr10.blocks[0];
        b[11] = e->iFile.addr10.blocks[1];
    }

    for(int i=0,j=0;i<fnumber,j<a;i++,j=j+4){
        if((j+4)>=(a-1)){
            this->Map[b[i]].isFree = false;
            this->Map[b[i]].fileName = e->fileName;
            this->Map[b[i]].data = data.substr(j,a-j);
            this->disk[b[i]] = data.substr(j,a-j);
```

```
        }
        else{
            this->Map[b[i]].isFree = false;
            this->Map[b[i]].fileName = e->fileName;
            this->Map[b[i]].data = data.substr(j,4);
            this->disk[b[i]] = data.substr(j,4);
        }
    }
    return STATUS_OK;
}
```

这一段代码首先调用了 indexFile 方法（见 1.2.1）获得了索引块，并将它赋给 FCB 控制块的指针，然后获取该索引块索引的所有盘块号（包括一级索引块的那些盘块号），接着统一进行位示图的初始化操作，给位示图赋值占用、文件名、数据内容等，并给磁盘块数据数组也进行数据内容的赋值，然后返回 STATUS_OK 标志，以表示目录系统请求的外存空间申请成功。

当目录系统申请读取文件内容时，便调用 receiveF_read 方法，磁盘管理系统会根据传入的 FCB 参数找到它在磁盘块中的所有索引，以及它们指向的数据块，然后将数据块内的数据取出，拼接进一个字符串内，最后将这个拼接成的字符串返回作为文件的数据内容，相关代码如下。

```
string DiskManager::receiveF_read(FCB* e){
    int fnumber = e->fileSize;
    int b[11] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
    for(int i=0;i<10;i++){
        b[i] = e->iFile.addr[i];
    }
    if(fnumber==11)
    {
        b[10] = e->iFile.addr10.blocks[0];
    }else if(fnumber>11)
    {
        b[10] = e->iFile.addr10.blocks[0];
        b[11] = e->iFile.addr10.blocks[1];
    }
    string a;
    for(int j=0;j<fnumber;j++){
        a.append(Map[b[j]].data);}
    return a;
}
```

1.3 设计结果

该模拟操作系统的可视化界面演示效果如下，首先生成了一个大小为 15 的文件。

图 1.3.1 生成数据

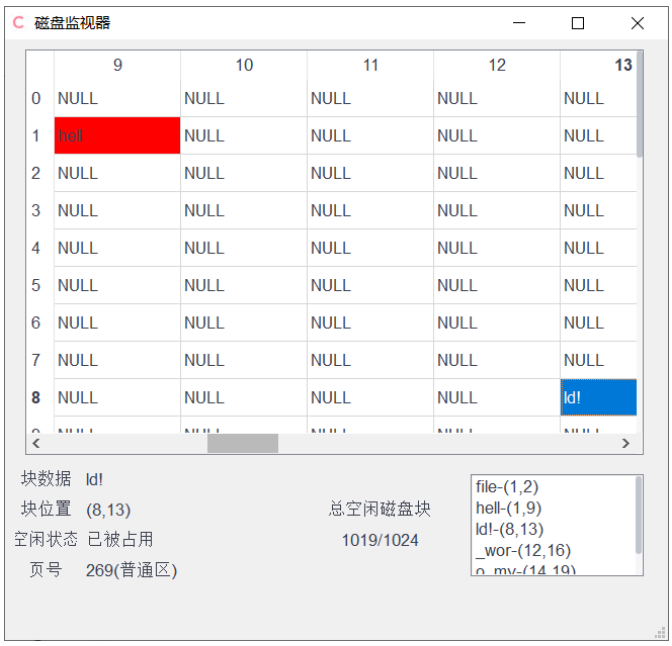

图 1.3.2 盘块分配情况 1

在生成文件后，文件的大小为 19，因此需要占用 5 个盘块，而总空闲磁盘块的数量变成了 1019，刚好占用了 5 个，说明程序运行结果是正确的，接着进行文件的删除操作，如图 1.3.3 所示。

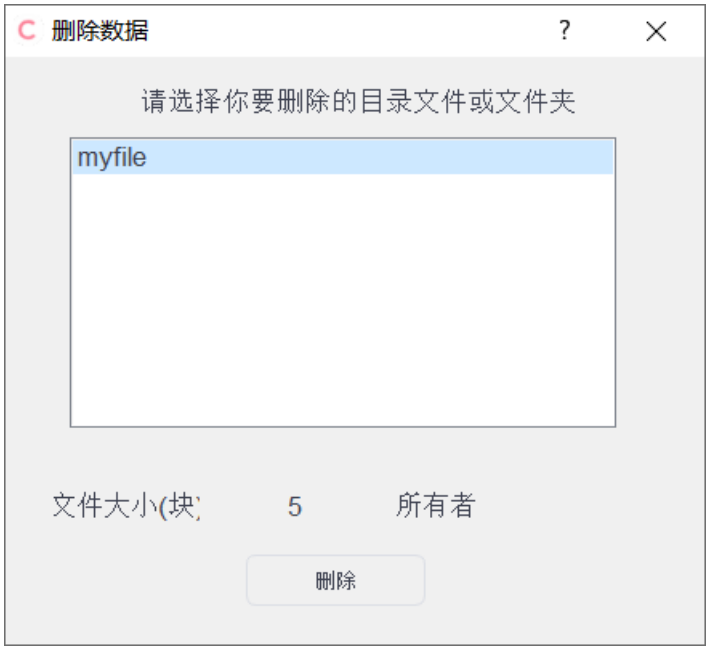

图 1.3.3 删除该项目

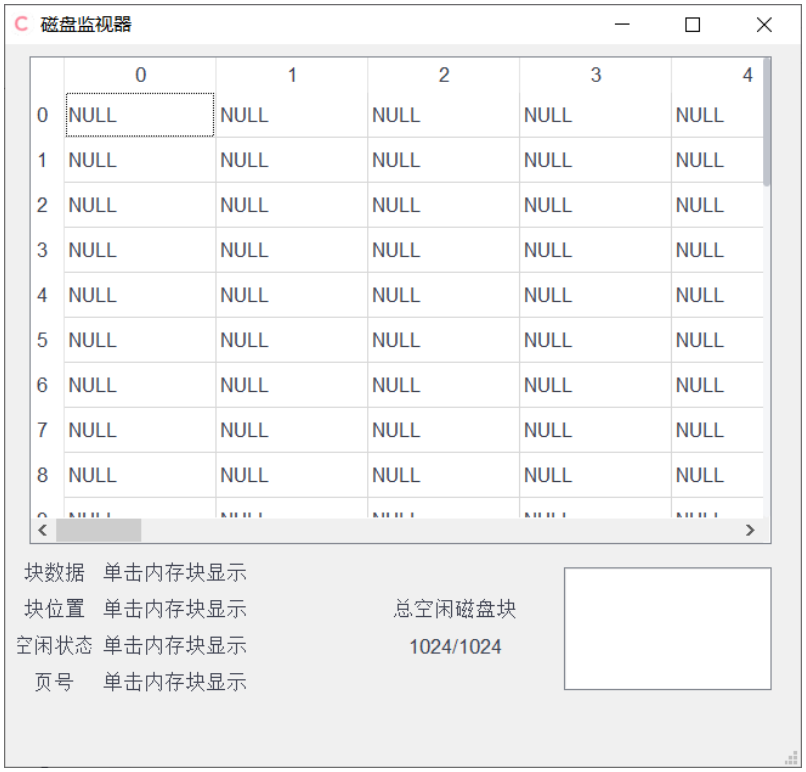

图 1.3.4 盘块分配情况 2

可以看到，在删除该文件后，总空闲磁盘块恢复为 1024，之前占用的盘块数被释放，成为空闲盘块。

1.4 设计结论

该模拟操作系统的外存管理部分能够成功实现盘块的分配和释放，以及构建出一级索引的管理模式，还能够实现外存对换区的换入和换出。因此该部分的基本功能得到了实现。

2 结束语

通过和其他同学的模块进行组装和接口的成功对接，本人完成了模拟操作系统中外存存储管理以及一级索引部分的设计。在设计的过程中，虽然遇到了许多困难，但在老师和同学的热情和悉心帮助下，以及互联网上大量资料的启发下，本人成功完成了这一模块代码的编写，并能够成功运行，实现了一级索引等功能，令本人感到很有成就感。

该模拟操作系统最后能够实现简单的操作系统基本操作，虽然实现简略，不过麻雀虽小，五脏俱全，其中的原理和技巧对个人的操作系统知识体系以及代码水平的挑战并不小。因此，通过本次操作系统课程设计，本人对操作系统相关知识点的认识得到了强化，同时，操作系统相关知识的学习不再局限于“纸上谈兵”，如同在空中楼阁一般，而是能够付诸于实践，在代码的编写和调试中得到更多对现代操作系统运行机理的领悟。

本次课程设计时间紧张，最后虽然能够做出实现了基本功能的成品，但本人认为代码本身还存在大量的不足之处，例如存在大量的冗余和无效信息，同时稍显繁琐的控制逻辑，都需要进一步得到优化。同时，和团队成员对接接口时，常常出现各种问题，这样导致代码更加复杂，因此本人意识到形成一套团队提交和合并代码规范流程体系的重要性，这应该也是分布式版本控制系统在软件工程界得到广泛应用的原因。

3 参考文献

[1] 梁红兵，哲凤屏, 汤小丹. 计算机操作系统[M]. 西安电子科技大学出版社, 2006.

[2] Remzi H. Arpaci-Dusseau,Andrea C. Arpaci-Dusseau. 操作系统导论[M].人民邮电出版社,2019.

[3] Tanenbaum A . 现代操作系统:原书第 3 版[M]. 机械工业出版社, 2010.

[4] 郑钢. 操作系统真象还原[M]. 人民邮电出版社, 2016.