实验1 存储管理

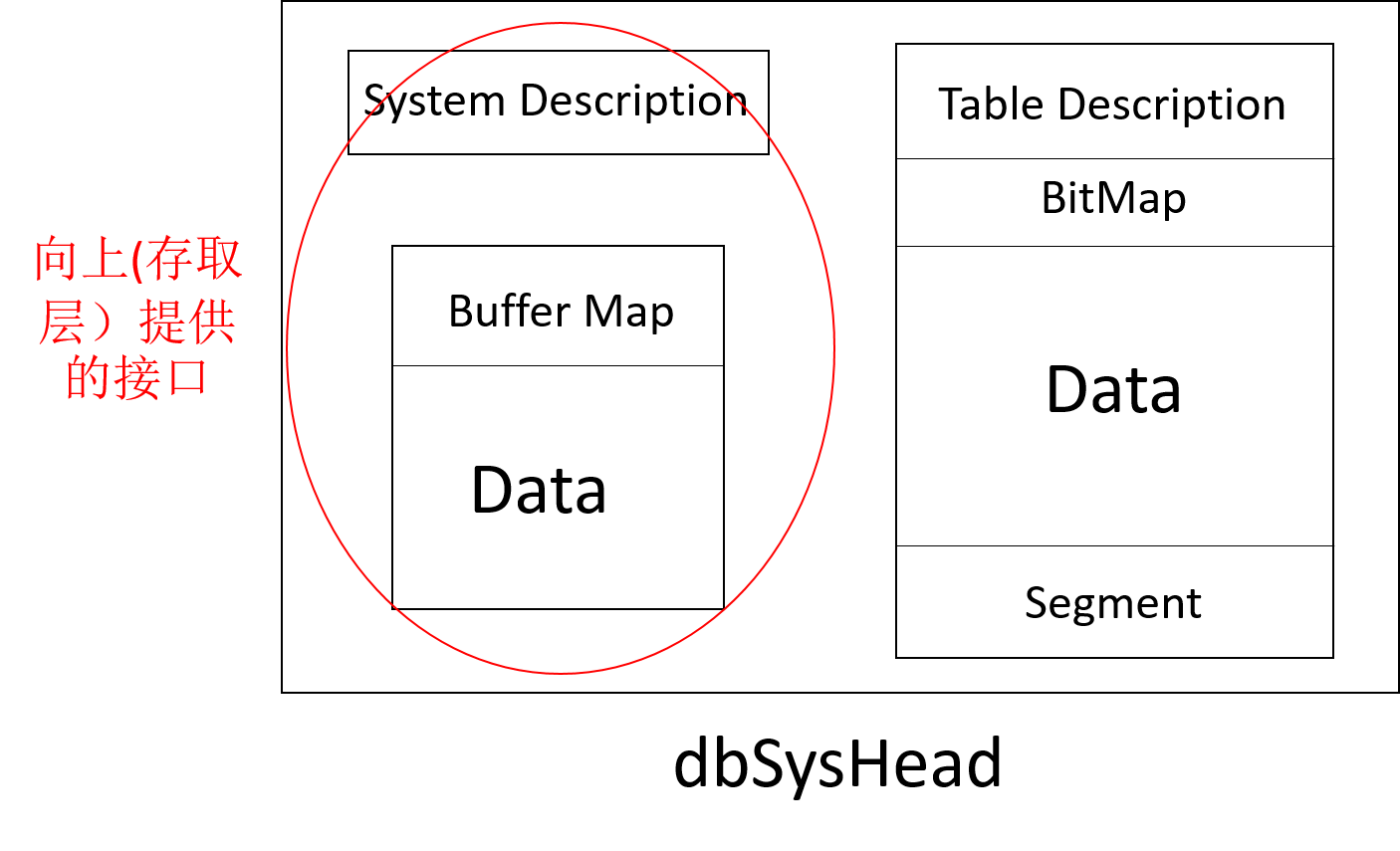
小组成员：吴天贞 陈师哲 周梦溪 蓝玮毓 张文慧

## 实验目的：

1. 数据文件的组织：段页式组织方式；支持基本数据类型
2. 缓冲区管理：缓冲区页面组织；缓冲区查找；缓冲区淘汰
3. 空闲空间管理：空闲空间的组织、分配和回收

## 实验设计

1. **整体框架**



1. **数据结构**

* 数据库系统：struct dbSysHead {

struct SysDesc desc; //数据库系统的元信息

struct buffSpace buff; //缓冲区对象

unsigned long \*bitmap; //描述数据库中各个页面是否为空

FILE \*fpdesc; //存储整个数据库系统的文件

}

* + 系统状态信息：struct SysDesc {

long sizeOfFile; // 数据库系统的容量

long sizePerPage; // 每一个页的大小

long totalPage; // 总共的页数

long pageAvai; // 当前有多少可用的页

long bitMapAddr; // bitMap在数据库系统中的起始地址

long sizeBitMap; // bitMap的大小，以字节为单位

long dataAddr; // 数据库系统中数据区的大小

long segmentNum; // 数据库系统中最多容纳段的数目

long segmentAddr; // 数据库系统中存储段的起始地址

long segmentCur; // 目前使用过的段的数量

long curfid; // 目前可以分配的fid号

long curFileNum; // 目前文件(表)的个数

struct FileDesc fileDesc[MAX\_FILE\_NUM]; //每一个文件的描述

};

* + 表文件信息：struct FileDesc {

long fileID; // 文件的标识

long fileAddr; // 文件的地址（Segment用链表串连）

long filePageNum; // 文件占用的页数

};

* + 段信息：struct Segment {

long fid; // 该Segment属于哪个文件

long count; // 该Segment中记录了多少个页

long nextAddr; // 下一个Segment的指针

long preAddr; // 前一个Segment的指针

long pageNo[PAGE\_PER\_SEGMENT]; // Page在数据库系统中的页号

};

* + 缓冲区页面信息：struct buffMap

{

long pageNo; // 该缓冲区块中存储的数据文件的页号

long loadTime; // 该页读入缓冲区的时间

long vstTime; // 访问该页的时间

int edit; // 该页中的数据是否被修改

};

* + 缓冲区信息：struct buffSpace {

char data[SIZE\_BUFF][SIZE\_PER\_PAGE]; // 缓冲区数据块

struct buffMap map[SIZE\_BUFF]; // 记录缓冲区块的信息

long curTimeStamp; // 目前的相对时间戳

};

1. **函数接口**
2. **缓冲区的管理**

|  |  |  |
| --- | --- | --- |
| 函数 | 输入 | 功能 |
| **reqPage** | **dbSysHead \*head,**  **long query** | **向缓冲区请求页面，返回其在缓冲区的下标。**   1. **如果请求的页面在缓冲区中，直接返回** 2. **如果请求的页面不在缓冲区中：** 3. **如果缓冲区未满，直接将页面调度近缓冲区** 4. **如果缓冲区已满，则使用页面调度算法淘汰缓冲区中的页，如果该页有修改过则需要写入文件中，再读入请求页面** |
| queryPage | head, query | 查询页面是否在缓冲区中 |
| replacePage | head, mapNo, pageNo | 用新的页面替换缓冲区中的原有的页面 |
| scheBuff | head | 缓冲区调度算法 |
| LRU | head | 淘汰最长时间没有读过或写过的缓冲区页面 |
| FIFO | head | 淘汰最先读进缓冲区中的页面 |

1. **内外存交换**

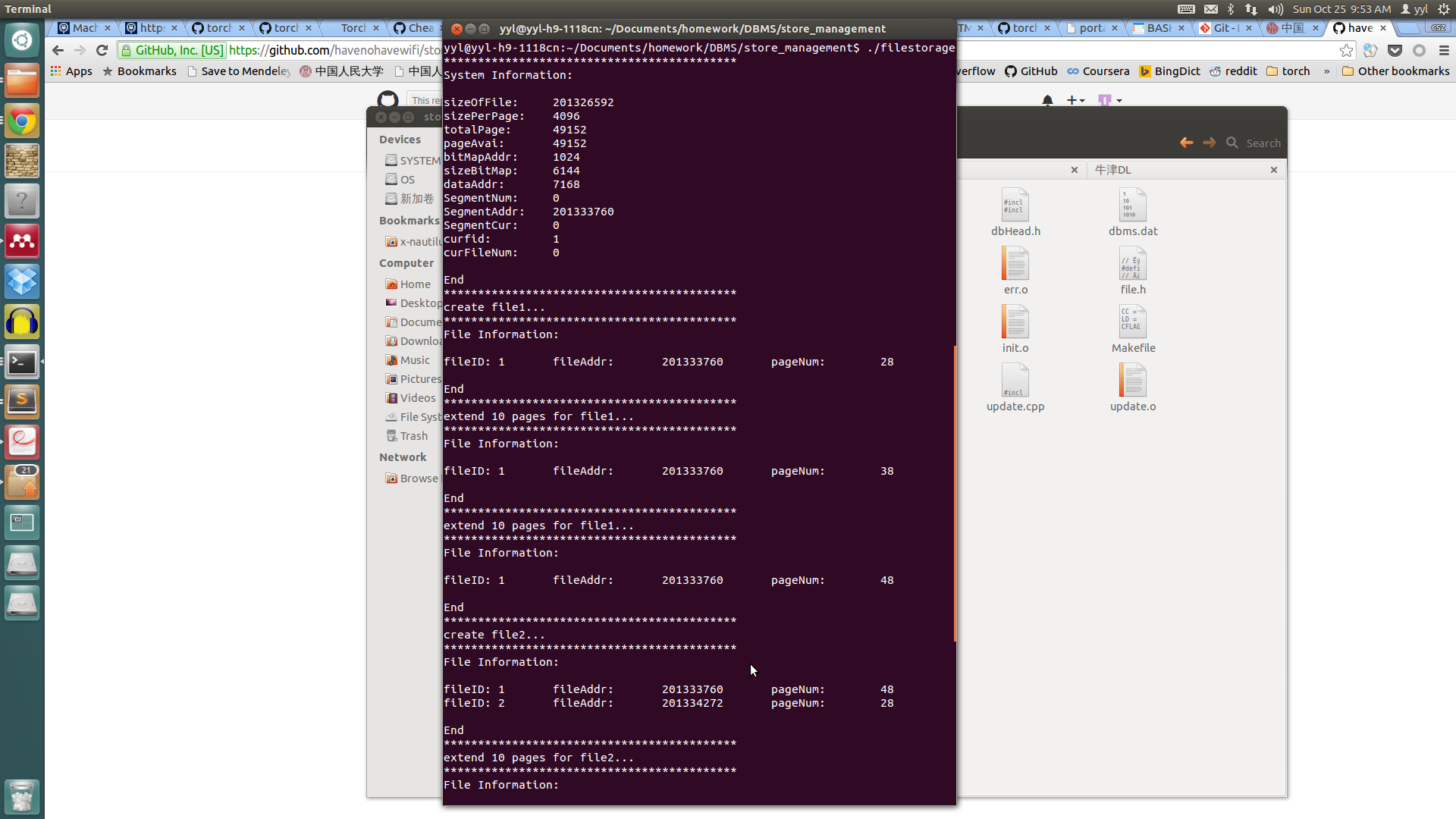
|  |  |  |
| --- | --- | --- |
| 函数 | 输入 | 功能 |
| **readFile** | **head, fid, pos, length, des** | **从表文件fid的任意位置pos读入长度为length的数据，写入des中。**   1. **表文件fid不存在，返回错误** 2. **否则，如果pos+length的位置大于表文件的最高位置，则溢出，返回错误** 3. **否则，读取数据所在的每一页** |
| **writeFile** | **head, fid, pos, length, des** | **向表文件fid中任意位置pos写入长度为length的数据des。（过程与读取类似）** |
| queryFileID | head, fid | 查询表文件fid是否存在 |
| manPage | head, fid, num | 把表文件中的页转化成数据库系统中的页 |
| readInPage | head, pgID, pos, length, des | 从数据库系统的页pgID中读出起始地址pos后的length长的数据，写入des中 |
| writeInPage | head, pgID, pos, length, des | 将des里的数据写入数据库系统的页pgID起始地址pos后的length长度中 |

1. **内存管理**

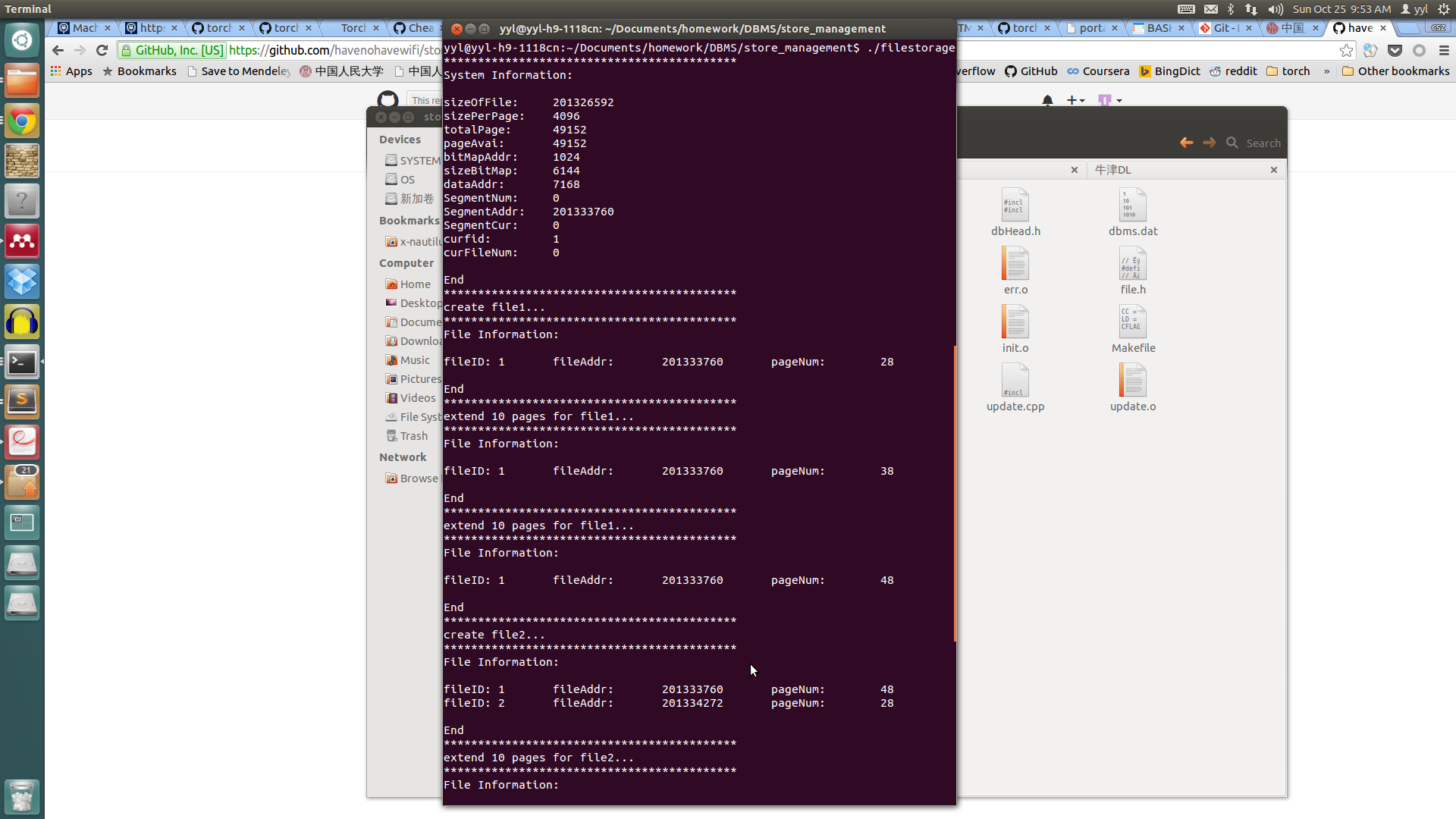
|  |  |  |
| --- | --- | --- |
| 函数 | 输入 | 功能 |
| initSegment | newSegment, startPage, pageNum, fd, preAddr, nextAddr | 初始化一个Segment：  startPage为属于这个Segment的页面起始地址，连续pageNum个页面均属于该Segment，该Segment属于fd表文件，并记录其上一个和下一个连接的块。 |
| getSegmentValue | head, segmentAddr, newSegment | 通过该Segment的地址从数据库系统中读取该Segment的值，保存到newSegment中。 |
| writeSegmentValue | head, segmentAddr, newSegment | 将newSegment的内容写入到数据库系统中segmentAddr位置上。 |
| GetLastSegmentAddr | head, fid | 获得表文件fid中最后一个Segment存储在数据库系统中的位置 |
| extendFileSpace | head, fid, extendPage | 拓展表文件fid的空间，要拓展的页数为extendPage |
| **createFileSpace** | **head** | **创建一个文件，并为其分配空间** |
| allocatePage | head, reqPageNum | 分配连续的空间，如果数据库中不存在reqPageNum连续长度的空间，则分配不成功，返回分配的起始地址 |
| recyOnePage | head, pageNo | 回收一个页 |
| recyPage | head, segmentAddr | 回收分配给一个表文件的所有页 |
| **recyFileSpace** | **head, fid** | **删除一个文件** |

## 结果展示

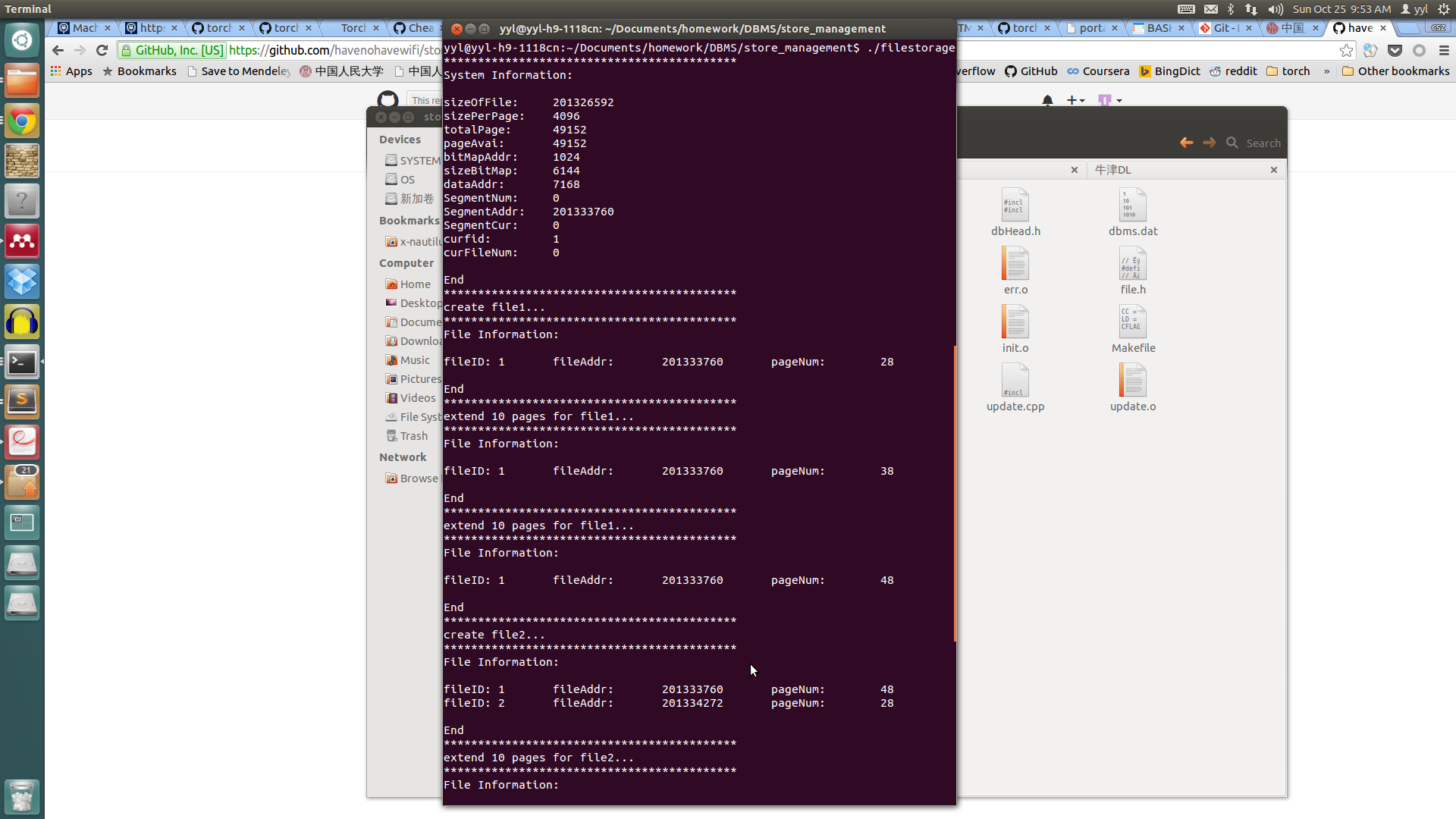
1. 创建和初始化数据库系统



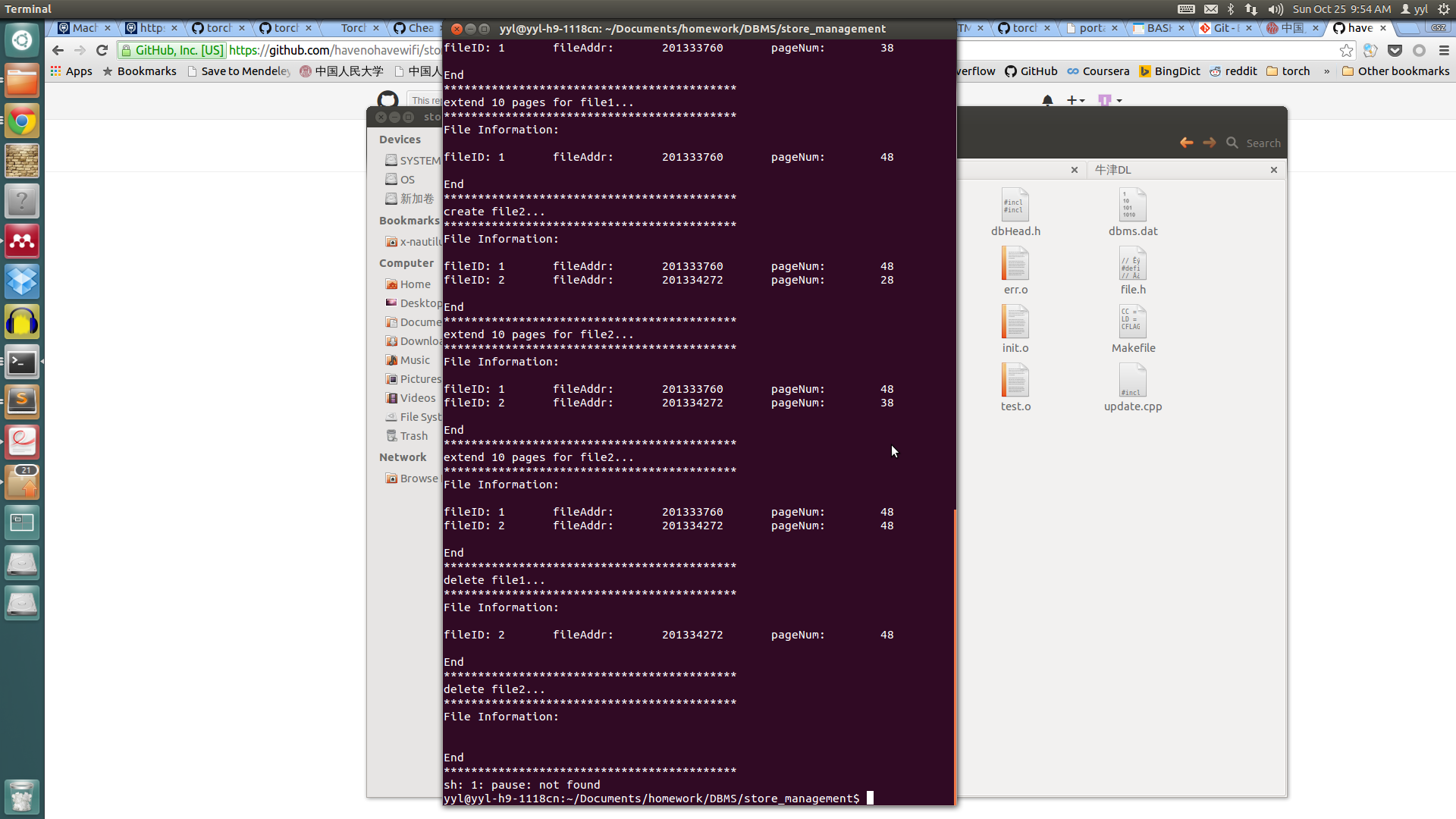
1. 创建一个新的表文件



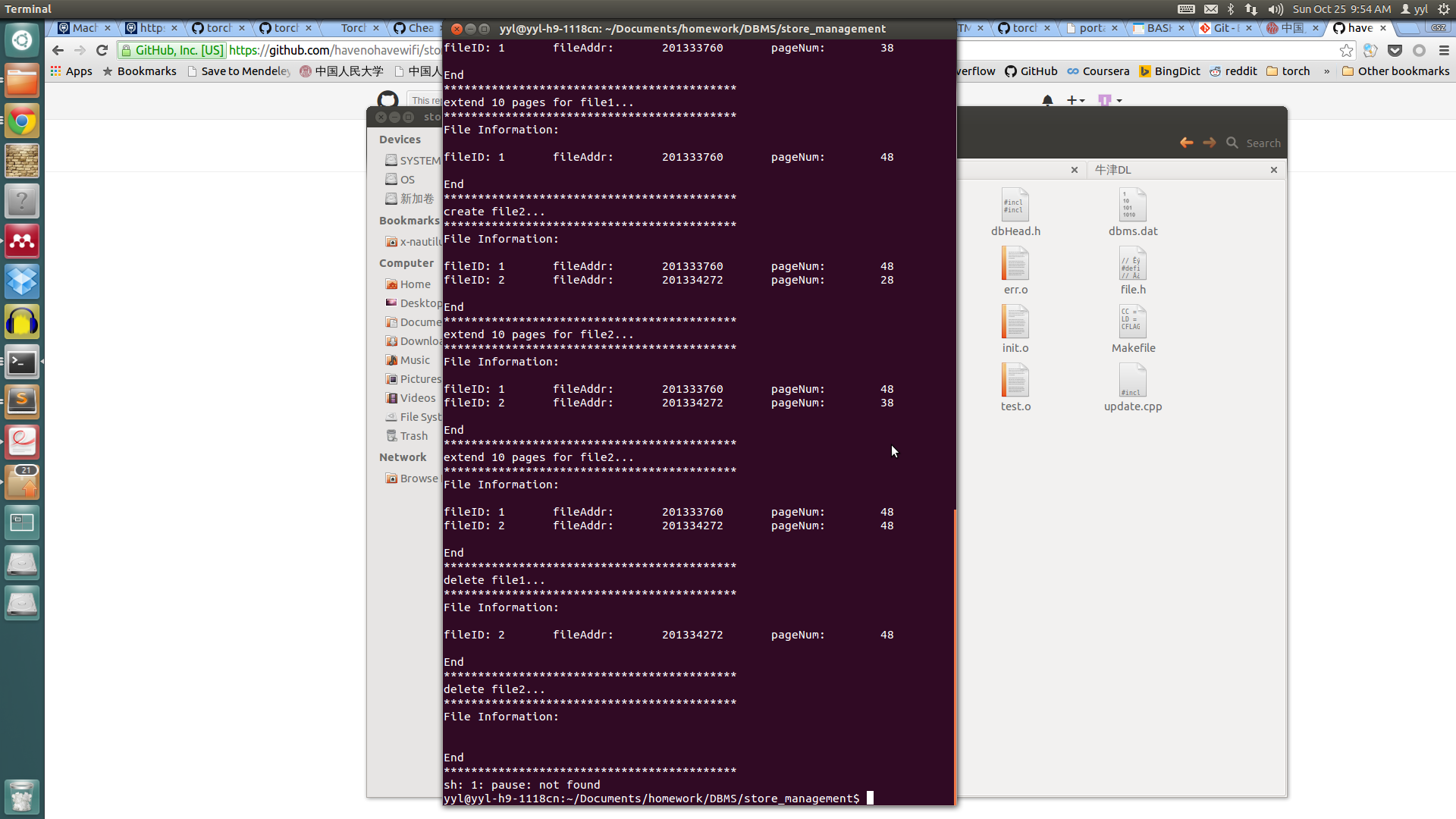
1. 拓展表文件的空间



1. 创建第二个表文件并拓展空间



1. 删除表文件



## 代码清单

见<https://github.com/havenohavewifi/store_management.git>