南京信息工程大学 实验（实习）报告

实验（实习）名称 UNIX磁盘空间管理方法：成组链接法 实验（实习）日期 得分 指导教师

系 计算机学院专业 物联网工程年级 2021 班次 2 姓名 学号

**一、实验目的**

设计磁盘文件系统时首先要解决磁盘存储空间的管理问题。作为一种现代计算机使用广泛的外存储设备，目前磁盘的存储空间越来越大，使用传统的空闲表法和空闲链接法进行管理的效率不高。对此，UNIX系统引入了成组链接法。成组链接法是一种优秀的磁盘空闲空间管理方法，具有效率高、不浪费磁盘存储空间的优点。本实验的目的是编程模拟实现磁盘成组链接方法的管理方式，加深实验者对该方法设计思想的理解和掌握。

本实验属于设计型实验，实验者可根据自身情况选用合适的开发环境和程序架构。

**二、实验原理与内容**

成组链接法是一种采用链栈来管理空闲存储空间的思想。

1. 内存中的空闲盘块号栈

系统在内存中准备一个栈，称空闲盘块号栈，此栈深度为100（或其他数目，读者自定），可以存储100个空闲盘块的块号，栈顶指针所指向的是当前可用的空闲盘块块号。空闲盘块号栈在多任务的UNIX环境中是一个临界资源，须互斥访问。

1. 磁盘上的空闲盘块号链

系统将所有空闲盘块的块号数据放在一个大的栈结构中，存放在磁盘上。由于此栈可能包含很多的盘块号，因此占用的存储空间会超过单个磁盘块的空间，需要将该栈分组存放，每组恰好放满一个磁盘块。组与组之间通过特别的方式链接起来，即前一组最末一个盘块号正好是下一组栈数据所存放的磁盘块号，构成空闲盘块号链。组的大小和内存中的空闲盘块号栈大小相同，保证空闲盘块号栈一次能放满一组数据。

1. 磁盘空间的分配和回收

系统初始化时，将空闲盘块号链的第一组数据调入内存中的空闲盘块号栈，调整栈指针到合适的位置，就可以进行磁盘块的分配和回收了。

* + 回收磁盘块

回收一个磁盘块时，将此磁盘块的块号入栈，栈顶指针向上移动一格。

* + 分配磁盘块

分配一个磁盘块时，将栈顶指针所指向的磁盘块块号出栈，分配该盘块给请求进程，栈顶指针向下移动一格。

* + 栈满处理

在空闲盘块号栈满时，如果再回收一个盘块，则首先将空闲盘块号栈中所有数据作为一组存放在该盘块中，情况内存的空闲盘块号栈，并将此盘块块号入栈。

* + 栈中仅有一个盘块

当栈中仅有一个盘块而又要再分配盘块时，首先将栈中最后一个盘块的内容读入空闲盘块号栈（因其中存放的就是下一组栈数据），然后分配该盘块。

操作系统关闭时，将空闲盘块号栈中的数据存入磁盘上的空闲块中。

编程实现上述的成组链接法，要求：

1. 在内存中开辟一片缓冲区，模拟磁盘的存储空间。缓冲区大小应为盘块大虾的整数倍。盘块大小根据用户自定，常见的有128字节，512字节和1K字节等。
2. 设计空闲盘块号栈的数据结构及常用的栈操作过程。请采用面向对象思想编程，用类来封装相应数据结构。栈中数据的变化应动态的显示在计算机屏幕上。
3. 编程模拟将分组数据写入磁盘和从磁盘读出的过程，即实现将分组数据写入缓冲区和从缓冲区读出。
4. 可以输入或动态产生一个磁盘的分配或回收请求。注意分配或回收时，可一次分配或回收若干盘块。
5. 将整个缓冲区所代表的磁盘空间以3盘块为单位显示在计算机屏幕上，动态地显示每次分配或回收盘块的结果。
6. **实验步骤**

在C语言中定义了以下的常量和类型：

#define BLOCK\_SIZE 128 *// 盘块的大小，单位为字节*

#define BLOCK\_NUM 8 *// 盘块的数量*

#define STACK\_SIZE 4 *// 栈的大小*

#define TRUE 1 *// 真值*

#define FALSE 0 *// 假值*

typedef int BOOL; *// 布尔类型*

typedef int BLOCK; *// 盘块类型，用整数表示盘块号*

typedef unsigned char BYTE; *// 字节类型*

空闲盘块号栈类的代码如下：

*// 定义一个空闲盘块号栈类*

typedef struct FreeBlockStack {

    int size; *// 栈的大小*

    int top; *// 栈顶指针*

    BLOCK stack[STACK\_SIZE]; *// 栈的存储空间，用数组表示*

} FreeBlockStack;

*// 初始化栈，设置栈的大小和栈顶指针*

void init\_stack(FreeBlockStack \*s, int size) {

    s->size = size; *// 设置栈的大小*

    s->top = -1; *// 设置栈顶指针为-1，表示栈为空*

}

*// 判断栈是否为空*

BOOL is\_empty(FreeBlockStack \*s) {

    return s->top == -1;

}

*// 判断栈是否为满*

BOOL is\_full(FreeBlockStack \*s) {

    return s->top == s->size - 1;

}

*// 入栈操作，将一个盘块号压入栈中*

void push(FreeBlockStack \*s, BLOCK block\_num) {

    if (is\_full(s)) { *// 如果栈满，打印错误信息*

        printf("Stack is full!\n");

    } else { *// 如果栈不满，栈顶指针加1，将盘块号存入栈中*

        s->top++;

        s->stack[s->top] = block\_num;

        printf("Push %d into stack.\n", block\_num); *// 打印入栈信息*

    }

}

*// 出栈操作，将一个盘块号弹出栈中*

BLOCK pop(FreeBlockStack \*s) {

    if (is\_empty(s)) { *// 如果栈空，打印错误信息*

        printf("Stack is empty!\n");

        return -1; *// 返回-1表示出错*

    } else { *// 如果栈不空，将栈顶的盘块号取出，栈顶指针减1*

        BLOCK block\_num = s->stack[s->top];

        s->top--;

        printf("Pop %d from stack.\n", block\_num); *// 打印出栈信息*

        return block\_num; *// 返回出栈的盘块号*

    }

}

*// 查看栈顶的盘块号，不改变栈的状态*

BLOCK peek(FreeBlockStack \*s) {

    if (is\_empty(s)) { *// 如果栈空，打印错误信息*

        printf("Stack is empty!\n");

        return -1; *// 返回-1表示出错*

    } else { *// 如果栈不空，返回栈顶的盘块号*

        return s->stack[s->top];

    }

}

*// 打印栈中的所有盘块号，从栈顶到栈底*

void print\_stack(FreeBlockStack \*s) {

    printf("Stack: ");

    for (int i = s->top; i >= 0; i--) {

        printf("%d ", s->stack[i]);

    }

    printf("\n");

}

分配和回收函数的代码如下：

*// 定义一个函数，用于分配一个盘块，返回分配的盘块号*

BLOCK allocate\_block(FreeBlockStack \*s, Disk \*d) {

    if (is\_empty(s)) { *// 如果栈空，表示需要从磁盘上读取一组空闲盘块号*

        if (s->top == -1) { *// 如果栈顶指针为-1，表示这是第一次分配，需要从磁盘的第0块开始读取*

            BLOCK block\_num = 0; *// 设置当前要读取的盘块号为0*

            BYTE \*data = read\_block(d, block\_num); *// 从磁盘读出第0块的数据*

            int count = \*(int \*)data; *// 从数据中获取该组的空闲盘块数目*

            for (int i = 0; i < count; i++) { *// 将该组的空闲盘块号依次压入栈中*

                block\_num = \*(BLOCK \*)(data + (i + 1) \* sizeof(BLOCK)); *// 获取第i+1个盘块号*

                push(s, block\_num); *// 将盘块号压入栈中*

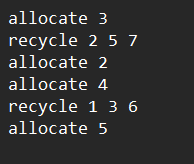
            }

            free(data); *// 释放数据的空间*

        } else { *// 如果栈顶指针不为-1，表示这不是第一次分配，需要从栈顶*

结果：

为了演示成组链接法的分配和回收过程，我假设您已经按照我给出的代码编译并运行了程序，并且在主函数中输入了以下的请求序列：



这些请求分别表示要分配3个盘块，回收盘块号为2，5，7的盘块，分配2个盘块，分配4个盘块，回收盘块号为1，3，6的盘块，以及分配5个盘块。

文本

描述已自动生成

缓冲区的变化如下（每行表示一个盘块，每个数字表示一个盘块号，每个箭头表示一个指针，每个空格表示一个空字节）：

文本

描述已自动生成