操作系统实验报告

实验（五）文件系统

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2023年6月8日
学　　院	信息学院
课程名称	操作系统

实验（五）文件系统

目录

1 实验目的

1. 学习掌握Linux系统中普通文件和目录文件的区别与联系
2. 学习掌握Linux管理文件的底层数据结构
3. 学习掌握Linux文件存储的常见形式
4. 加深对读写者问题的理解和信号量的使用

2 实验内容

2.1 changeDir函数

首先使用 findUnitInTable 函数定位目标目录项的索引。如果返回的索引为-1，则表示该目录项不存在，需要输出提示信息。接着，检查该目录项是否是目录类型，如果不是需要输出提示信息。若是目录，则需要修改当前目录，并根据需要修改全局绝对路径。具体来说，如果目标目录是父目录，则需要去掉路径最后一部分，否则需要在路径和目录名之间进行拼接，以得到新的路径。

```
int changeDir(char dirName[])
{

        int unitIndex = findUnitInTable(currentDirTable, dirName);

        if (unitIndex == -1)
        {
                printf("file not found\n");
                return -1;
        }

        if (currentDirTable->dirs[unitIndex].type == 1)
        {
                printf("not a dir\n");
                return -1;
        }

        int dirBlock = currentDirTable->dirs[unitIndex].startBlock;
        currentDirTable = (dirTable*)getBlockAddr(dirBlock);


        if (strcmp(dirName, "..") == 0)
        {

                int len = strlen(path);
                for (int i = len - 2; i >= 0; i--)
                        if (path[i] == '/')
                        {
                                path[i + 1] = '\0';
                                break;
                        }
```

```
        }
        else
        {

                strcat(path, dirName);
                strcat(path, "/");
        }
        return 0;
}
```

2.2 changeName函数

首先需要定位到要修改的目录项。如果文件名不存在，则需要输出提示信息。如果存在，则可以直接修改目录项的名称。

```
int changeName(char oldName[], char newName[])
{
        int unitIndex = findUnitInTable(currentDirTable, oldName);
        if (unitIndex == -1)
        {
                printf("file not found\n");
                return -1;
        }
        strcpy(currentDirTable->dirs[unitIndex].fileName, newName);
        return 0;
}
```

2.3 creatDir函数

首先需要验证目录名的长度是否合法。如果目录名过长，需要输出提示信息。接着，可以使用 getBlock 函数为目录表分配一个大小为 1 的空间。如果分配失败，则需要输出提示信息。然后需要使用 addDirUnit 函数将新目录作为目录项添加至当前目录。按照文件系统的规定，还需要向父目录添加一个目录项。我们可以创建一个新的目录项表，其中需要将目录项数目初始化为 0。最后需要再次调用 addDirUnit 函数将父目录添加到新目录表，并将父目录设置为当前目录。

```
int creatDir(char dirName[])
{
        if (strlen(dirName) >= FILENAME_MAX_LENGTH)
        {
                printf("file name too long\n");
                return -1;
        }

        int dirBlock = getBlock(1);
        if (dirBlock == -1)
                return -1;

        if (addDirUnit(currentDirTable, dirName, 0, dirBlock) == -1)
                return -1;
```

```
        dirTable* newTable = (dirTable*)getBlockAddr(dirBlock);
        newTable->dirUnitAmount = 0;
        char parent[] = "..";
        if (addDirUnit(newTable, parent, 0, getAddrBlock((char*)currentDirTable)) == -1)
                return -1;
        return 0;
}
```

2.4 addDirUnit函数

首先需要对目录项的合法性进行检测。具体分为两个部分：一是检查目录是否已满，二是遍历目录表检查是否存在同名文件。如果目录项不合法，则添加失败，并显示相应提示信息。如果目录项合法，则构建新目录项，并设置 fileName 文件名、type 类型和 FCBBlockNum 大小等各个属性。

```
int addDirUnit(dirTable* myDirTable, char fileName[], int type, int FCBdataStartBlock)
{

        int dirUnitAmount = myDirTable->dirUnitAmount;

        if (dirUnitAmount == DIR_TABLE_MAX_SIZE)
        {
                printf("dirTables is full, try to delete some file\n");
                return -1;
        }

        if (findUnitInTable(myDirTable, fileName) != -1)
        {
                printf("file already exist\n");
                return -1;
        }

        dirUnit* newDirUnit = &myDirTable->dirs[dirUnitAmount];
        myDirTable->dirUnitAmount++;

        strcpy(newDirUnit->fileName, fileName);
        newDirUnit->type = type;
        newDirUnit->startBlock = FCBdataStartBlock;
        return 0;
}
```

2.5 deleteFile函数

根据文件系统的规定，自动创建的父目录不能被删除，需要忽略掉。接着要查找目标文件的索引。如果查找失败或者查找到的目录项的文件类型是目录，则删除失败，并显示相应提示信息。否则需要释放该文件的内存并将该目录项从目录表中删除。

```
int deleteFile(char fileName[])
{
```

```
        if (strcmp(fileName, "..") == 0)
        {
                printf("can't delete ..\n");
                return -1;
        }

        int unitIndex = findUnitInTable(currentDirTable, fileName);
        if (unitIndex == -1)
        {
                printf("file not found\n");
                return -1;
        }
        dirUnit myUnit = currentDirTable->dirs[unitIndex];

        if (myUnit.type == 0)
        {
                printf("not a file\n");
                return -1;
        }
        int FCBBlock = myUnit.startBlock;

        releaseFile(FCBBlock);

        deleteDirUnit(currentDirTable, unitIndex);
        return 0;
}
```

2.6 write_file函数

该函数与 read_file 函数一起构成读者写者问题。 write_file 函数比较简单，只需要考虑写者锁，而不需要考虑读者锁。首先需要获取文件的控制块，接着获取写者锁。如果有写者正在写，则需要等待。在获取了写者锁之后，就可以开始写操作。首先获取文件大小和要写入数据的大小。需要注意的是，文件的大小是块的数量乘以块大小的乘积，不能仅将块的数量作为文件大小。然后使用 getBlockAddr 函数获取写入位置，进行循环写操作。写操作完成后，需要释放写者锁，以让其他写者写入数据。此外，需要检测写入的数据大小是否等于文件大小，如果相等，则表示文件已满，输出提示信息。

```
int write_file(char fileName[], char content[])
{

        FCB* myFCB = open(fileName);
        int contentLen = strlen(content);
        int fileSize = myFCB->fileSize * block_size;
        char* data = (char*)getBlockAddr(myFCB->dataStartBlock);
        myFCB->write_sem = sem_open("write_sem", 0);

        if (sem_wait(myFCB->write_sem) == -1)
                perror("sem_wait error");

        for (int i = 0; i < contentLen && myFCB->dataSize < fileSize; i++, myFCB->dataSize++)
```

```
        {
                *(data + myFCB->dataSize) = content[i];
        }

        printf("> Write finished, press any key to continue....");
        getchar();

        sem_post(myFCB->write_sem);
        if (myFCB->dataSize == fileSize)
                printf("file is full, can't write in\n");
        return 0;
}
```

3 实验总结

本次实验中，我成功地实现了一个类似于 ramfs 的内存文件系统 myRAMFS。通过这个实验，我对文件系统底层存储数据结构的理解得到了加深，并且学习了如何实现一个简单的内存文件系统。在实验中，我通过学习掌握Linux系统中普通文件和目录文件的区别与联系，更深入地理解了Linux管理文件所使用的底层数据结构。通过阅读、理解提供的文件并正确处理文件系统的数据结构，包括目录项、文件节点等，我能够轻松地完成接下来的实验任务。通过类比已经实现的函数，加上注释的指导，我分别实现 changeDir、changeName、creatDir、addDirUnit、deleteFile 和 write_file 函数。其中 write_file 函数难度稍大，需要处理读写者问题和信号量的使用，联系了之前学习的内容。最后，我逐步测试了每个功能的正确性。

在实验过程中，我通过实际操作实现了文件系统的创建、删除和文件的写入、读取等基本操作，并且可以支持多个进程同时对文件进行读写操作，更深入地体会到了信号量的使用和读写者问题的解决方法。我们还深入学习了Linux文件存储的常见形式，并且复习了学期初学习的如何使用信号量解决读写者问题。通过本次实验，我不仅巩固了对Linux文件系统底层数据结构的理解，还复习了同步互斥的各种方法，同时增强了对Linux系统编程的能力。总的来说，这个实验不仅对于加深我对文件系统的理解非常有帮助，还提高了我的编程技能和问题解决能力。