上海大学

SHANGHAI UNIVERSITY

操作系统（二）实验报告

组　　号	第 6 组
学号姓名	20121034 胡才郁
实验序号	实验 4
日　　期	2023 年 2 月 27 日

一　实验目的与要求

目的

随着社会信息量的极大增长，要求计算机处理的信息与日俱增，涉及到社会生活的各个方面。因此，文件管理是操作系统的一个极为重要的组成部分。学生应独立地用高级语言编写和调试一个简单的文件系统，模拟文件管理的工作过程。从而对各种文件操作命令的实质内容和执行过程有比较深入的了解，掌握它们的实施方法，加深理解课堂上讲授过的知识。

要求

1. 实际一个 n 个用户的文件系统，每个用户最多可保存 m 个文件。
2. 限制用户在一次运行中只能打开 1 个文件。
3. 系统应能检查打入命令的正确性，出错要能显示出错原因。
4. 对文件必须设置保护措施，如只能执行，允许读、允许写等。在每次打开文件时，根据本次打开的要求，再次设置保护级别，即可有二级保护。
5. 对文件的操作至少应有下述几条命令：
    - delete 删除文件。
    - open 打开文件。
    - close 关闭文件。
    - read 读文件。
    - write 写文件。

二　实验环境

- 操作系统：macOS
- 处理器架构：ARM

其余主要实验环境配置如图 1所示。

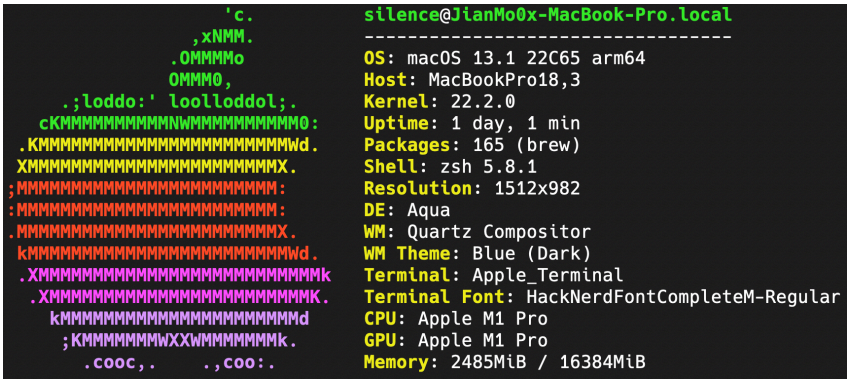

图 1: 实验环境

三　实验内容及其设计与实现

1　数据结构设计

在本次实验中，文件系统的多几分层可以分为以下几个层次：

- MFD
- UFD
- AFD
- FILE

其中，具体实现采用 C++ 语言中的 STL 基本库中的常用容器作为数据结构，如 vector、map 等。

代码 1: MFD 结构设计

```
map<string, UFD> mfd;
```

对于 MFD 而言，使用 Map 容器作为数据结构，存放 Pair 的键值对。其中 key 为用户名，value 为 UFD 结构。使用 Map 容器作为数据结构，利用了 Map 中键是唯一的特点，即在二级文件目录结构下，所有用户名都是唯一的。因此使用 Map 作为容器很好的实现了 MFD 的功能，即可以通过用户名快速查找到对应的 UFD 结构。

代码 2: UFD 结构设计

```
struct UFD {
    vector<File> files;
    AFD afd;
};

struct AFD {
    vector<File> opened_files;
};
```

对于 UFD 而言，使用 vector 容器作为数据结构，存放 File 结构。其中 File 结构包含文件名、文件大小、文件保护级别等信息。vector 容器为动态数组，实现容量扩充操作比较方便。对于文件查找操作而言，需要遍历 vector 下的所有文件，时间复杂度为 $O(n)$。其中的 AFD 为 Files 的 vector 容器，存放当前用户打开的文件，通过判断 open_files 的长度即可动态的控制用户打开文件的数量。

代码 3: AFD 结构设计

```
struct File {
    string filename;
    int filesize;
    string property;
};
```

设置 File 结构体，包含文件名、文件大小、文件保护级别等信息。此处不拘泥于实验指导书中使用三个二进制位来表示读写修改权限，而是使用 UNIX 操作系统中的”rwx” 字符串来表示文件的保护级别。

2 核心功能设计

2.1 用户信息认证

本次实验中的文件系统是多用户的，因此需要对用户进行认证，即用户在使用文件系统之前，需要先输入用户名，系统会根据用户名查找对应的 UFD 结构。如果找到了对应的 UFD 结构，则认证成功，否则认证失败。认证成功后，用户可以对该用户的文件进行操作。认证失败后，用户需要重新输入用户名，直到认证成功为止。

MFD 设计为 Map 容器，其中的 pair 键值对中，key 为用户名，value 为 UFD 结构。因此可以通过用户名快速查找到对应的 UFD 结构。在 STL 中，map 是用红黑树实现的，因此查找的时间复杂度为 $O(logn)$，比 vector 容器的 $O(n)$ 要快很多。用户信息认证的代码实现如**代码 4**所示，其中 find 函数返回的是一个迭代器，如果迭代器不等于 mfd.end()，则表示找到了对应的 UFD 结构，认证成功，并取出其对应的 UFD 结构。

代码 4: AFD 结构设计

```
while (true) {
  cout << "YOUR NAME ?" << endl;
  string username;
  cin >> username;
  if (auto search = mfd.find(username); search != mfd.end()) {
      UFD ufd = search->second;
      break;
  }
  cout << "YOUR NAME IS NOT N THE USER NAME TABLE,TRY AGAIN." << endl;
}
```

2.2 列出文件

对于列出某用户文件夹下的所有文件，需要遍历该用户的 UFD 结构中的 vector 容器，打印出所有文件的信息。其中，文件信息包括文件名、文件大小、文件保护级别等。他的功能类似于 Linux 操作系统之中的`ls`指令，如**代码 5**所示。

代码 5: 列出文件

```
void ls(UFD ufd) {
  cout << "File Name\t Protection\t File Size" << endl;
  for (auto file: ufd.files) {
      cout << file.filename << "              \t" << file.property << "
          \t" << file.filesize << endl;
  }
  cout << endl;
}
```

2.3 删除文件

删除文件、读写文件等操作，其离不开的就是文件的查找。在本次实验中，文件查找的时间复杂度为 $O(n)$，因此在实现删除文件的功能时，需要遍历 vector 容器，找到对应的文件，然后删除。在双层文件系统之中，对于每一个用户级别的目录下，要求的文件不可以重名，因此在遍历 vector 容器时，只需

要找到对应的文件名即可。核心实现如**代码 6**所示。erase 函数的参数为迭代器，因此需要将 vector 容器的下标转换为迭代器，即`ufd.files.begin() + i`。

代码 6: 列出文件

```
for (int i = 0; i < ufd.files.size(); i++) {
  if (ufd.files[i].filename == filename) {
      ufd.files.erase(ufd.files.begin() + i);
      cout << "Delete a file successfully!" << endl;
      break;
  }
}
```

四　实验结果

进入主界面后，输入已经存储过的用户名称，会打印该用户已有的文件信息，包括文件名、文件大小、文件权限等。如**图 2**所示。

```
YOUR NAME ?
usr_a
File Name    Protection  File Size
a               r              1
b               rw             2
c               r              3

Input a command:
1. Create a file
2. Delete a file
3. Read a file
4. Write a file
5. Open a file
6. Close a file
7. List all files
8. Exit
```

图 2: 主功能界面

当选择创建文件功能后，输入文件名、文件大小、文件权限，即可创建文件，创建前判断用户当前已有文件数量是否超过用户保存的最大文件数量（实验设置为 10），即通过判断 vector 容器的长度判断数量是否超过最大值。若未超过，则在 vector 容器中添加文件信息，否则提示用户文件数量已达上限。如**图 3**所示。

```
Input a command:
1. Create a file
2. Delete a file
3. Read a file
4. Write a file
5. Open a file
6. Close a file
7. List all files
8. Exit
1
Input a filename: create_test
Input a filesize: 5
Input a property: rwx
Create a file successfully!
```

图 3: 创建文件

对于读写功能而言，首先判断用户是否有读写权限，若有，则读写文件，否则提示用户无权限。此处试图写文件 c，而文件 c 不包含“w”权限，因此提示用户无权限。如图 4所示。

```
File Name    Protection  File Size
a               r               1
b               rw              2
c               r               3
Input a filename: c
You don't have the permission to write this file!
```

图 4: 写文件权限不足

五　收获与体会

Linux 中的文件结构是一个树形结构，每个文件夹都是一个节点，文件夹下的文件也是一个节点，文件夹下的文件夹也是一个节点，这样就构成了一个树形结构。在本次实验中，我将文件夹看作是一个用户，用户下的文件夹看作是一个文件，用户下的文件看作是一个文件。这样就构成了一个双层文件系统，用户之间的文件夹是独立的，用户之间的文件也是独立的。在实现文件夹的创建、删除、读写等功能时，需要遍历 vector 容器，找到对应的文件夹或文件，然后进行相应的操作。在实现文件夹的创建、删除、读写等功能时，需要遍历 vector 容器，找到对应的文件夹或文件，然后进行相应的操作。

在本次实验中完成了二级目录结构的编写，我对 C++ 语言的 STL 库有了更深的了解，也对 C++ 语言的一些特性有了更深的了解。操作系统中的许多算法都离不开基本数据结构，如链表、哈希表等等，这些数据结构的使用在本次实验中得到了很好的锻炼。