**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **操作系统** |
| **学生姓名：** | **葛子午** |
| **学生学号：** | **202330550381** |
| **学生专业：** | **软件学院** |
| **开课学期：** | **2024-2025年度第二学期** |

**华南理工大学软件学院**

**2025年6月**

**目 录**

**[实验一：Linux 常用命令使用及C程序编写 2](#_Toc198808450)**

**[实验二：进程创建及进程间通信 15](#_Toc198808451)**

**[实验三：进程调度算法模拟实验 25](#_Toc198808452)**

**[实验四：存储管理实验 37](#_Toc198808453)**

**[实验五：简单文件模拟系统 50](#_Toc198808454)**

**实验一：Linux 常用命令使用及C程序编写**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7 楼 | 331房； | **评 分：** |  |
| **实验日期与时间：** | 2025.5.8 | | **实验教师：** | 王国华 |

## **一、实验目的**

1． 了解 Linux 一般命令格式。

2． 掌握常用命令和选项的功能。

3． 学习使用 vi 编辑器建立、编辑、显示及加工处理文本文件。

4． 掌握 Linux 下 C 程序的编写、编译与运行方法。

## **二、实验环境**

**1. 操作系统**：Ubuntu

2. 工具

终端（Terminal），vi/vim 文本编辑器，gcc 编译器

## **三、实验内容**

1. 熟练使用man命令及date等简单命令。

2. 在用户主目录下对文件进行操作：复制一个文件、显示文件内容、查找指定内容、排序、文件比较、文件删除等。

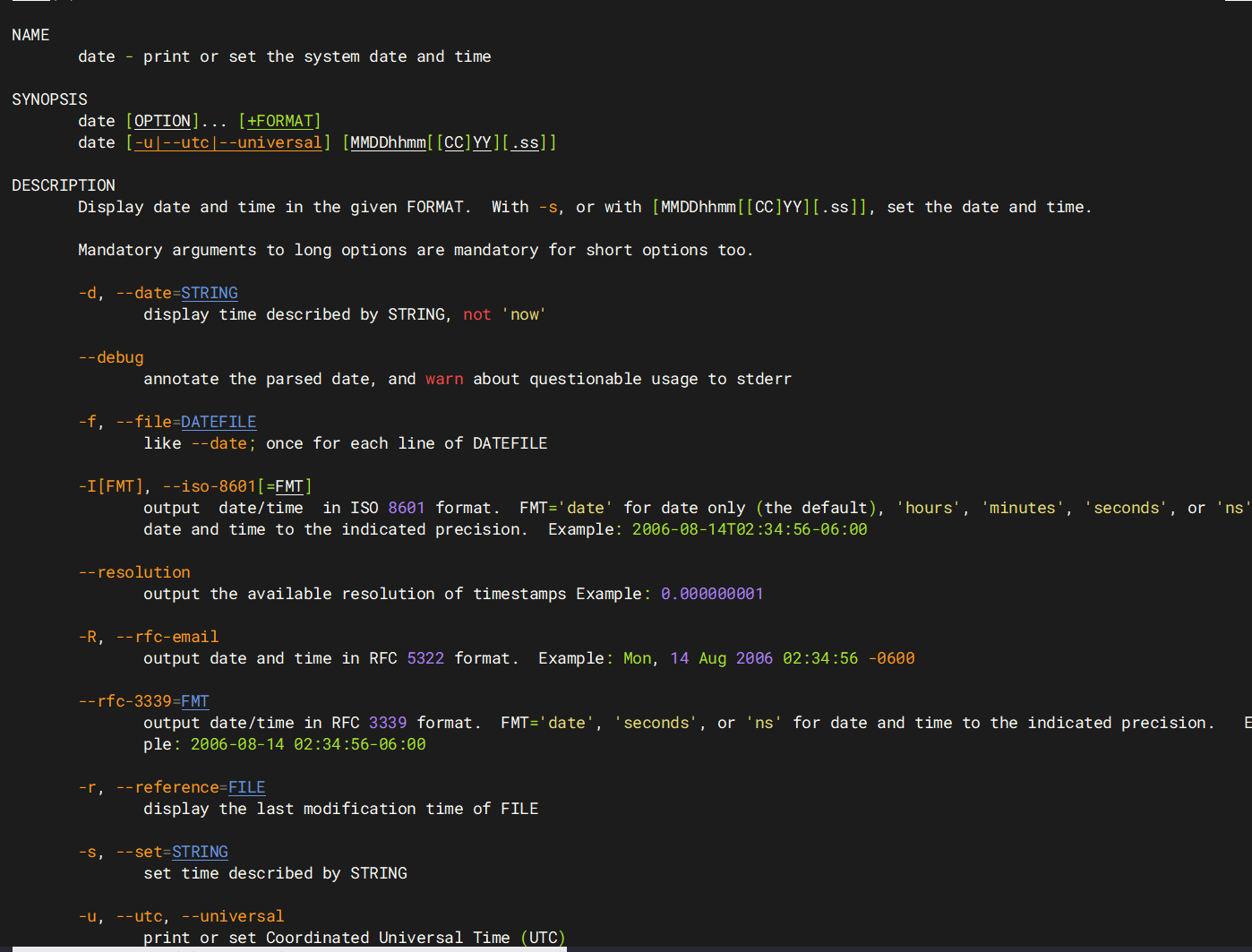
3. 对目录进行管理：创建和删除子目录、改变和显示工作目录、列出和更改文件权限、链接文件等。

4. 使用vi 建立一个文本文件，并在该文件上移动光标位置，编写任意文件内容，然后对该文件执行删除、复原、修改、替换等操作。

5. 用vi编辑器编写一个简单的C程序，用gcc编译器编译该程序并执行。

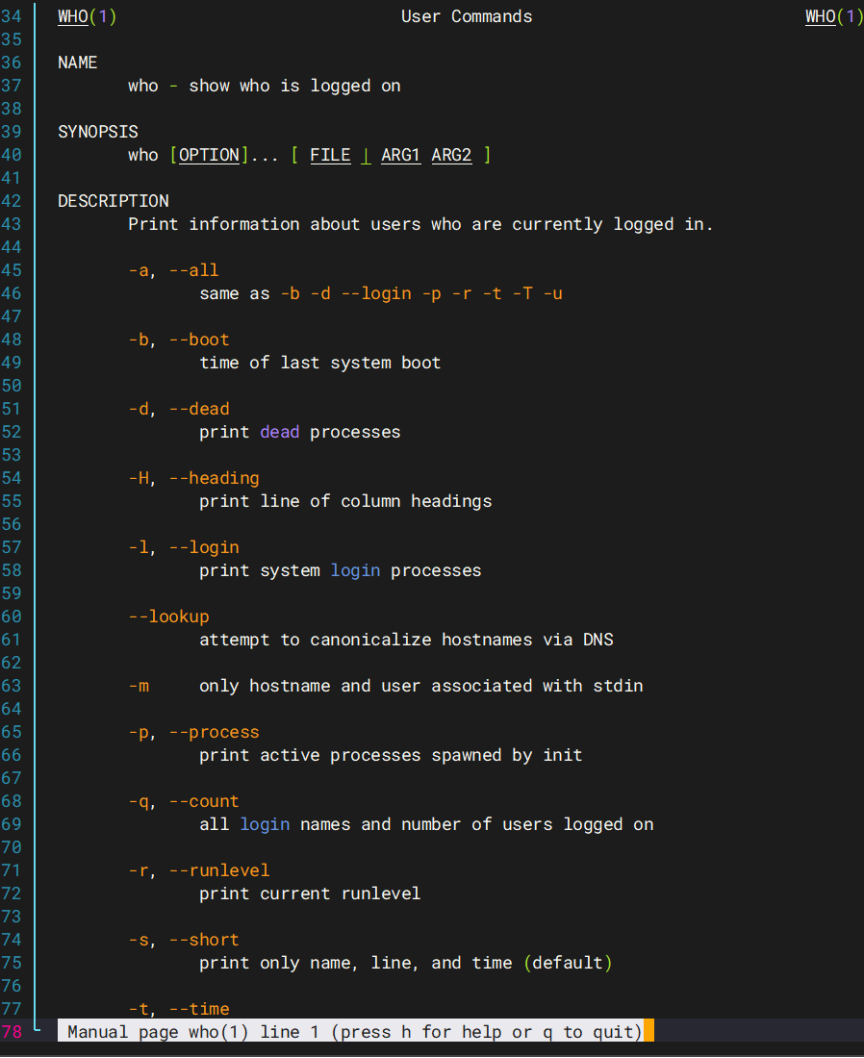
## **实验步骤**

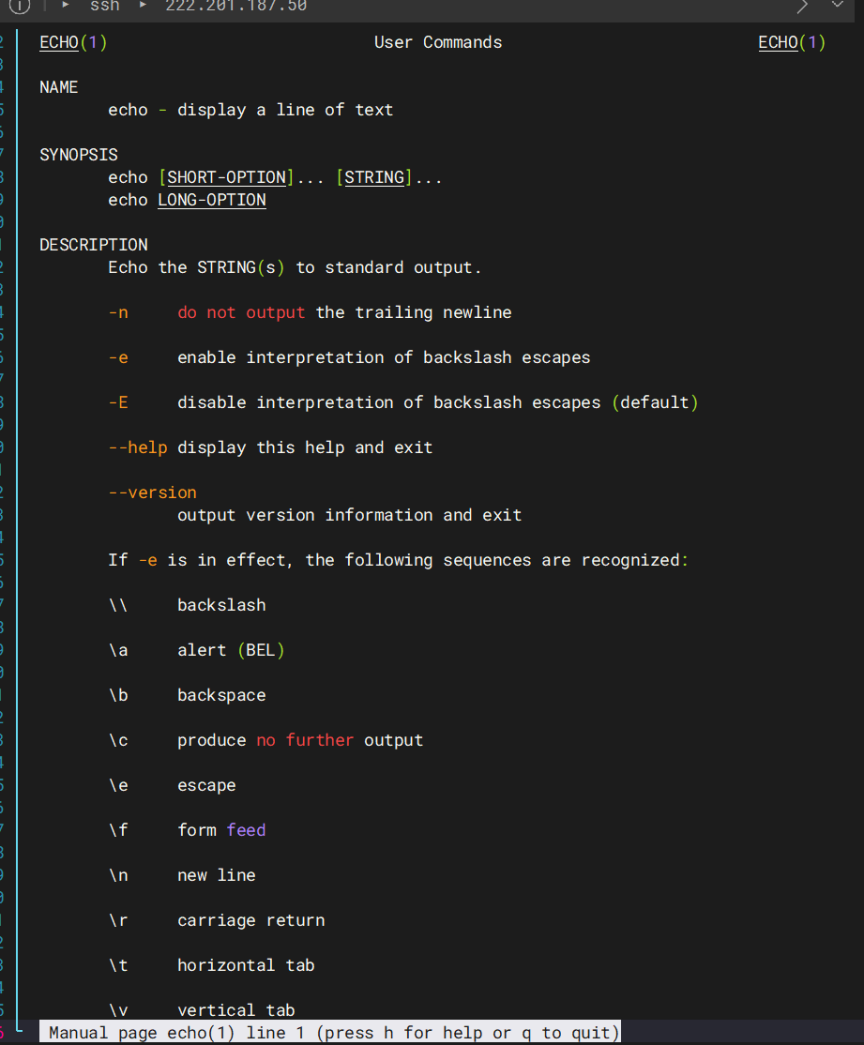
1. 使用man显示date等命令的手册页。



2. 通过使用简单命令date, cal, who, echo, clear等，了解Linux命令格式。

如图所示



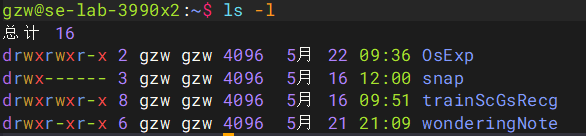


3. 浏览文件系统

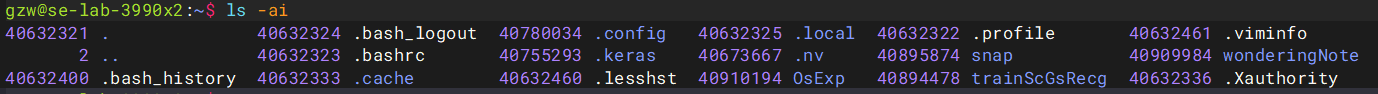
(1) 运行pwd命令，确定当前工作目录。



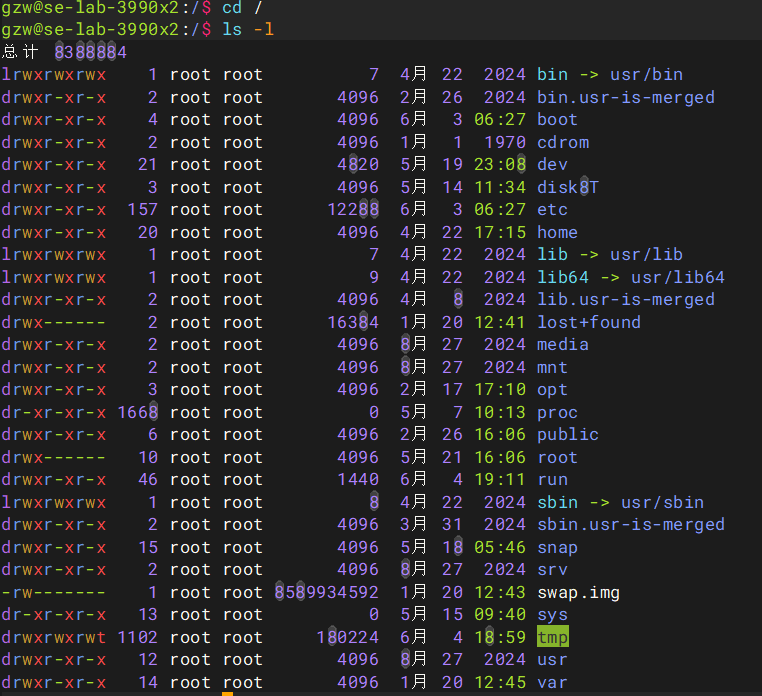
(2) 运行ls –l命令，理解各字段含义。



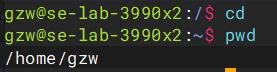
1. 运行ls –ai命令，理解各字段含义。



1. 使用cd命令，将工作目录改为根目录（/）。运行ls –l命令，了解各目录的作用。



(5) 直接使用cd，回到哪里了？用pwd验证。



(6) 用mkdir建立一个子目录subdir。

(7) 将工作目录改到subdir。



4. 文件操作

(1) 验证当前工作目录在subdir。



1. 运行date > file1，然后运行cat file1，看到什么信息？

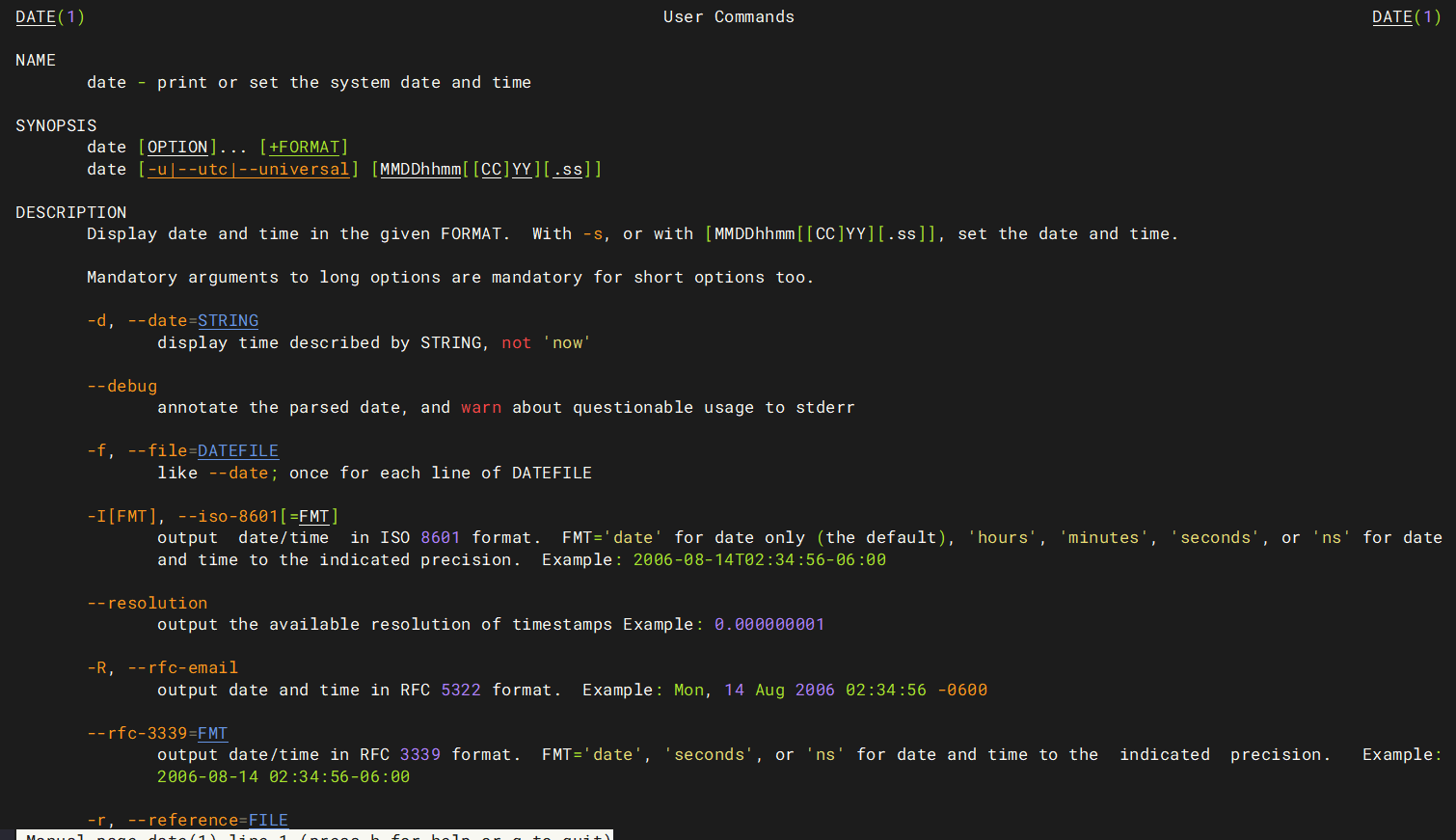


(3) 运行cat subdir，会有什么结果？为什么？

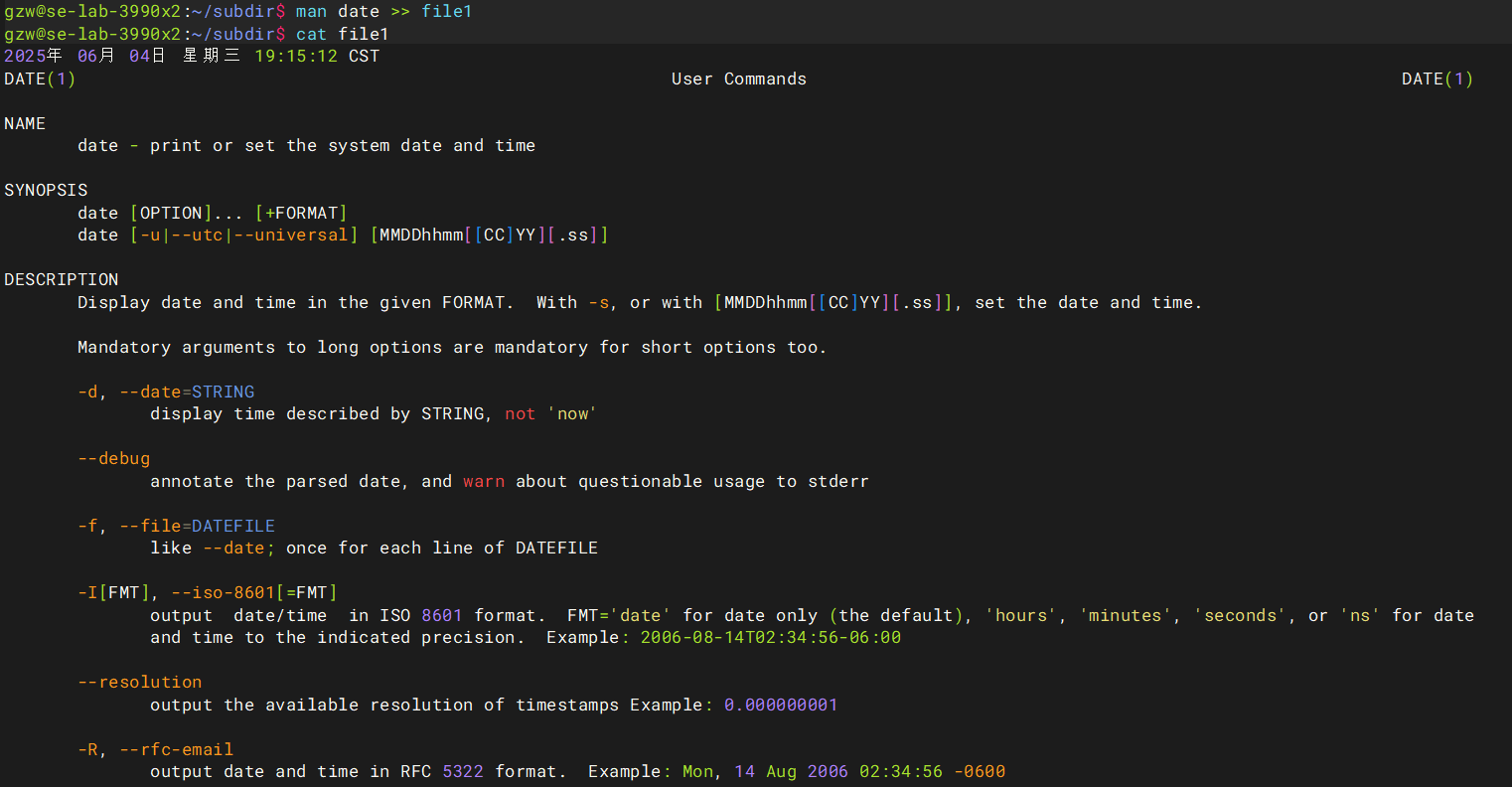


subdir是目录而非文件，cat命令只能查看文件内容。应使用ls subdir查看目录内容。

(4) 利用man命令显示date命令的使用说明。

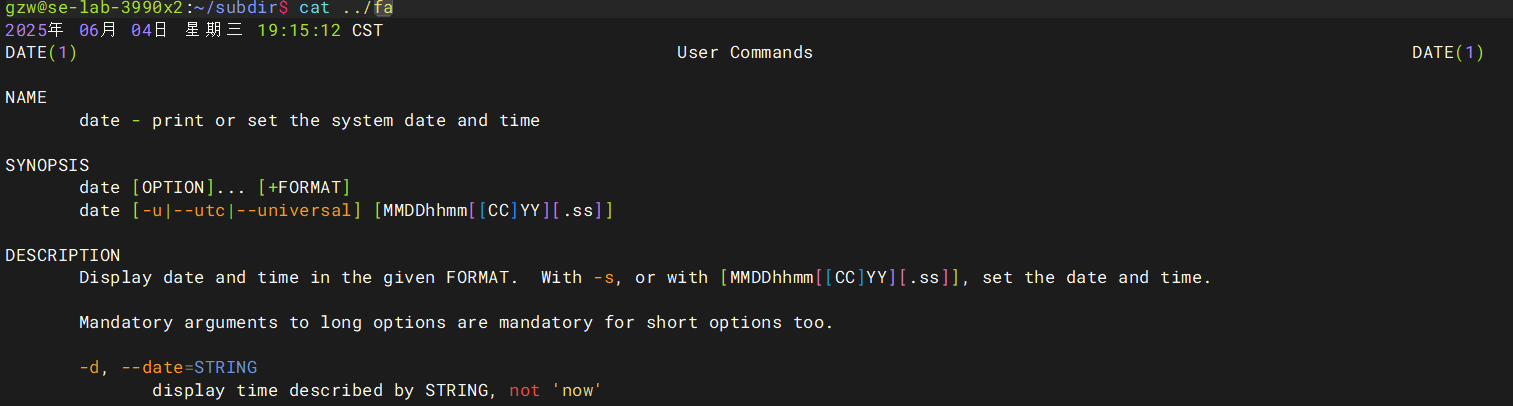


(5) 运行man date >>file1，看到什么？运行cat file1，看到什么？

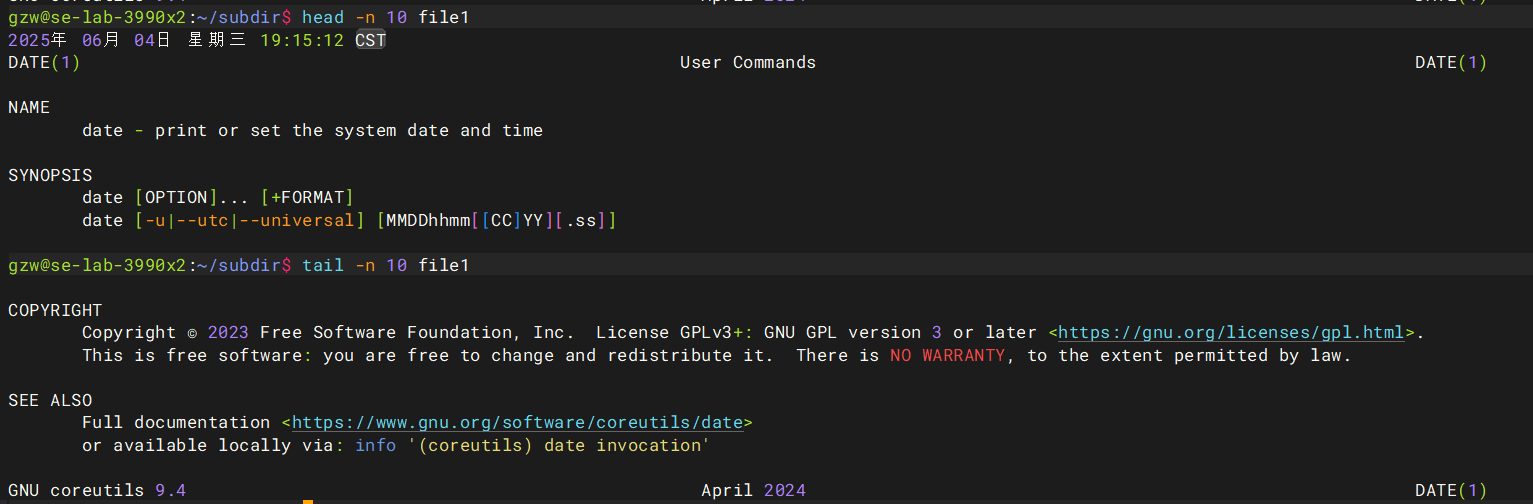
(6) 利用ls –l file1，了解链接计数是多少？运行ln file1 ../fa，再运行ls –l file1，看链接计数有无变化？用cat命令显示fa文件内容。

有变化，如图。硬链接与原文件共享inode，删除原文件仍可访问；软链接是独立文件，存储目标路径，原文件删除后失效。



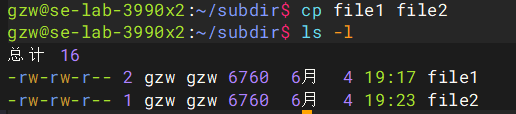


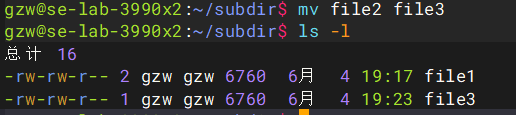
1. 显示file1的前10行，后10行。

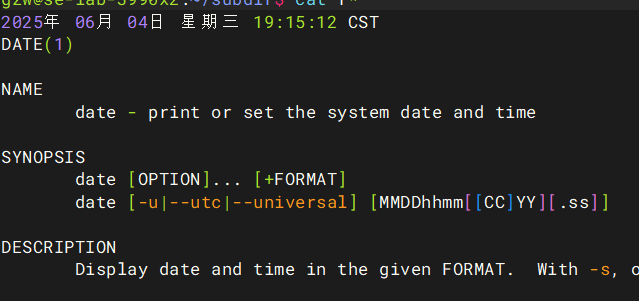


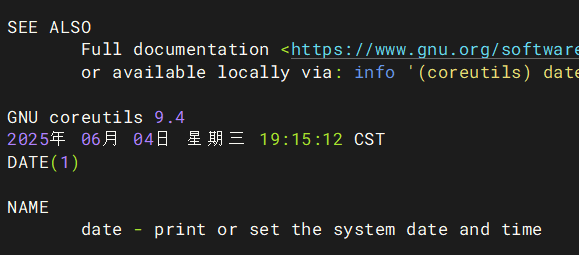
1. 运行cp file1 file2，然后ls –l，看到什么？运行mv file2 file3，然后ls –l，看到什么？运行cat f\*，结果怎样？

cat f\*显示了file1、file3的内容

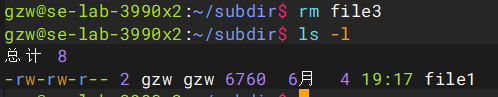




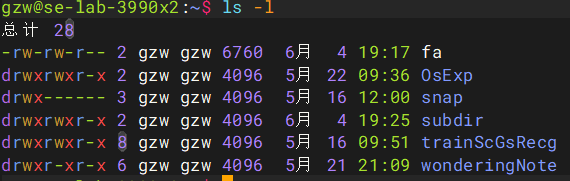




(9) 运行rm file3，然后ls –l，结果如何？



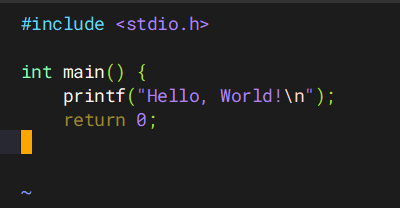
1. 运行ls –l，理解各文件的权限是什么？



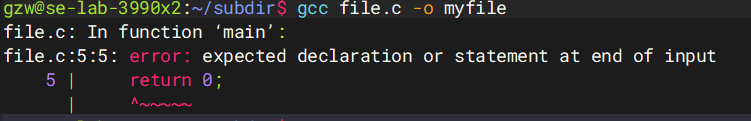
运行 ls -l 命令后，每个文件的权限信息由 10 个字符组成。第一个字符表示文件类型（如 - 是普通文件，d 是目录）。后 9 个字符分为三组，分别表示所有者、所属组和其他用户的权限，每组依次是读（r）、写（w）、执行（x）权限，- 表示无权限。例如 -rwxr-xr-- 表示：这是一个普通文件，所有者可读、写、执行，组用户可读和执行，其他用户仅可读。权限也可以用数字表示（如 755），其中每位数字是 r(4)、w(2)、x(1) 的和。

5. vi 编辑器

(1) 建立一个文件，如file.c。进入插入方式，输入一个C语言程序的各行内容，故意制造几处错误。最后，将该文件存盘。回到shell状态下。

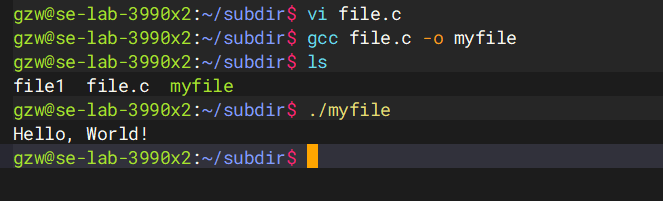


1. 运行gcc file.c –o myfile，编译该文件，会发现错误提示。理解其含义。



1. 重新进入vi，对该文件进行修改。然后存盘，退出vi。重新编译该文件。如果编译通过了，可以用./myfile运行该程序。

修改后的结果



1. 运行man date > file10，然后vi file10。使用x，dd等命令删除某些文本行。使用u命令复原此前的情况。

删除以及复原操作

x删除单个字符，dd删除整行，u复原



## **五、实验总结**

通过本次实验，系统掌握了Linux基础命令的操作逻辑与vi编辑器的核心工作模式，能够熟练使用ls -l分析文件权限结构（如-rwxr-xr--对应的用户/组权限），并通过gcc编译调试纠正了C程序中的语法错误与逻辑错误。实验过程中发现，硬链接与软链接在实际操作中的行为差异（如删除源文件后的访问性）验证了文件系统的inode机制，而管道命令的重定向特性为后续进程通信实验奠定了基础。

实验二：进程创建及进程间通信

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7 楼 | 331房； | **评 分：** |  |
| **实验日期与时间：** | 2025.5.08 | | **实验教师：** | 王国华 |

## **一、实验目的**

1. 掌握Linux 进程的创建方法,加深对进程概念的理解,明确进程和程序的区 别｡

2. 认识进程并发执行的实质｡

3. 学习控制进程同步的方法｡

4. 分析进程竞争资源的现象,学习解决进程互斥的方法｡

5. 了解管道通信的特点,掌握管道通信的使用方法｡

## **二、实验环境**

1. 操作系统：Linux（Ubuntu）

2. 工具：

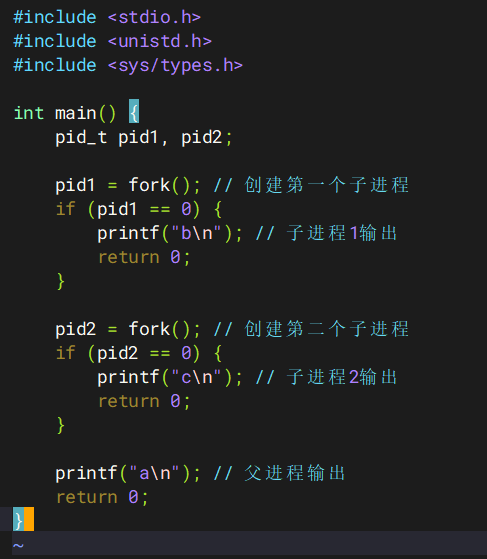
gcc 编译器

终端（Terminal）

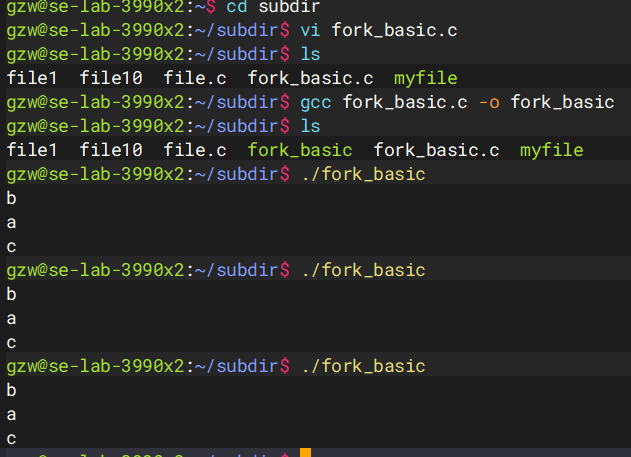
## **三、实验内容及代码**

1. 进程的创建｡编写一段C/C++程序,使用系统功能调用fork( )创建两个子进 程｡当此程序运行时,在系统中有一个父进程和两个子进程活动｡让每一个 进程在屏幕上显示一个字符;父进程显示字符“a”,子进程分别显示字符 “b”和“c”｡试观察记录屏幕上的显示结果,并分析原因｡

代码如下：



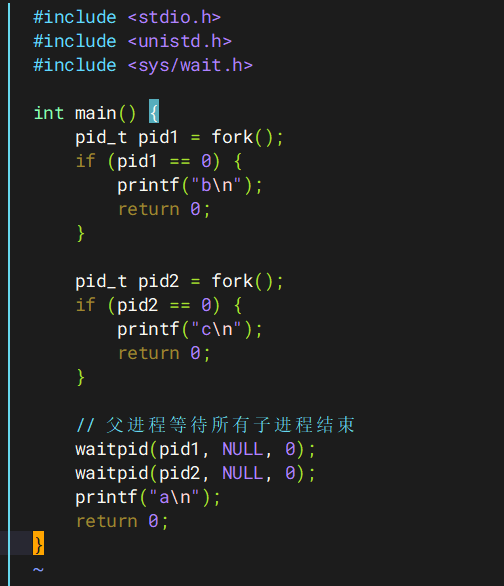
运行结果：



该程序的输出顺序（如 a b c 或 b a c 等）是不确定的，因为 fork() 创建的子进程与父进程并发执行，由操作系统调度决定运行顺序；此外，若未在 fork() 前刷新标准输出缓冲区（fflush(stdout)），可能导致父进程的 printf 缓冲数据被子进程继承，造成重复输出（如 a a b c）。这体现了进程执行的异步性和缓冲机制对结果的影响。

2. 修改1.的程序,要求子进程先输出,然后父进程再输出｡

代码如下：

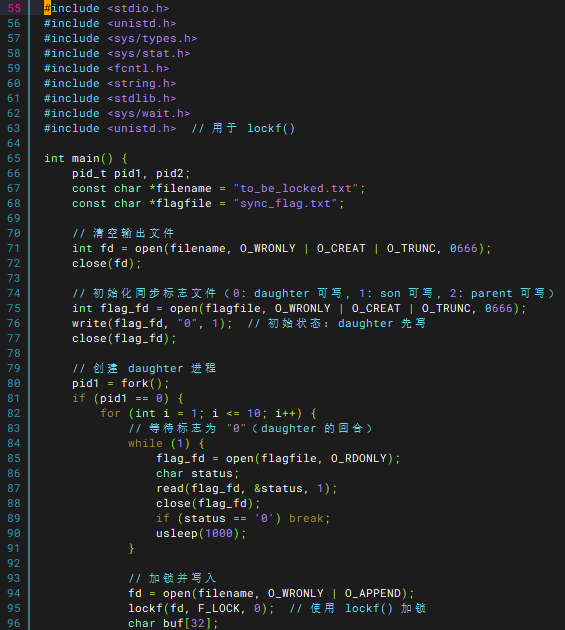


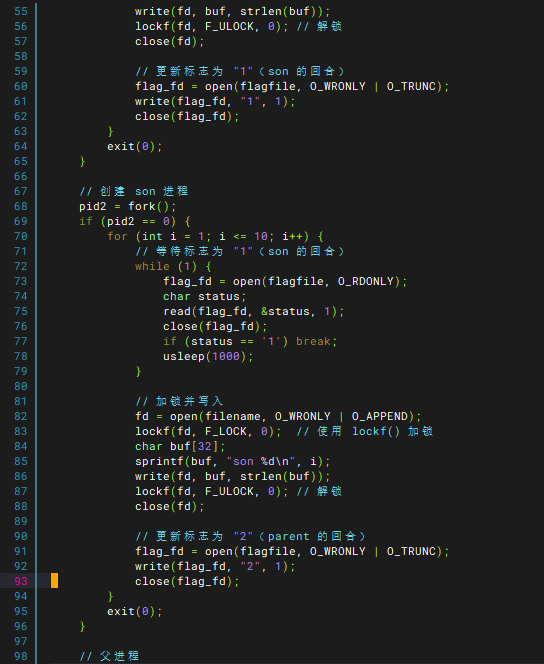
运行结果：

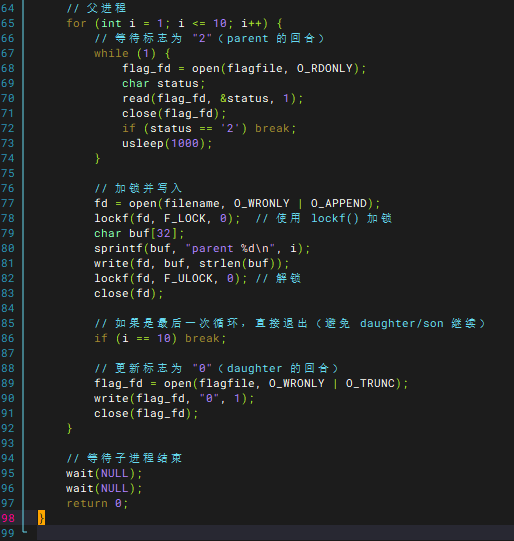


3. 修改1.的程序,每一个进程循环十次,每次循环:子进程1 将“daughter i” 输出到文件to\_be\_locked.txt 文件中,子进程2 将“son i”输出到文件 to\_be\_locked.txt 文件中,父进程将“parent i”写入到文件to\_be\_locked.txt 文件中｡用lockf( )来给每一个进程加锁,以实现进程之间的互斥｡使用catto\_be\_locked.txt 查看输出结果,并分析原因｡

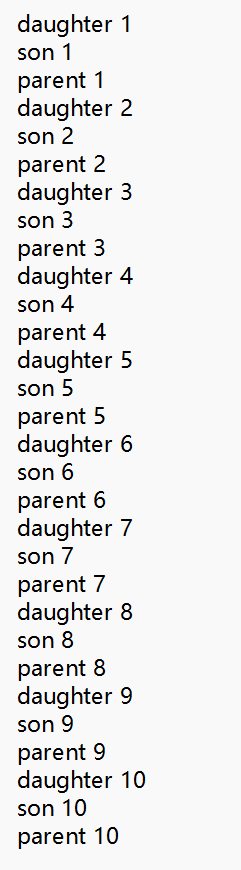
代码如下：







运行结果：



使用 lockf() 加锁后，文件 to\_be\_locked.txt 的输出结果会严格按daughter i、son i、parent i 的顺序交替出现，因为 lockf() 实现了进程互斥，确保同一时刻只有一个进程能写入文件，避免了数据竞争。若不加锁，输出可能会交错混乱（如 daughter 1, parent 1, son 1...），甚至出现行内容混合（如 daughtparent 1er 1），而加锁保证了每个进程的完整写入，使结果有序且可预测。

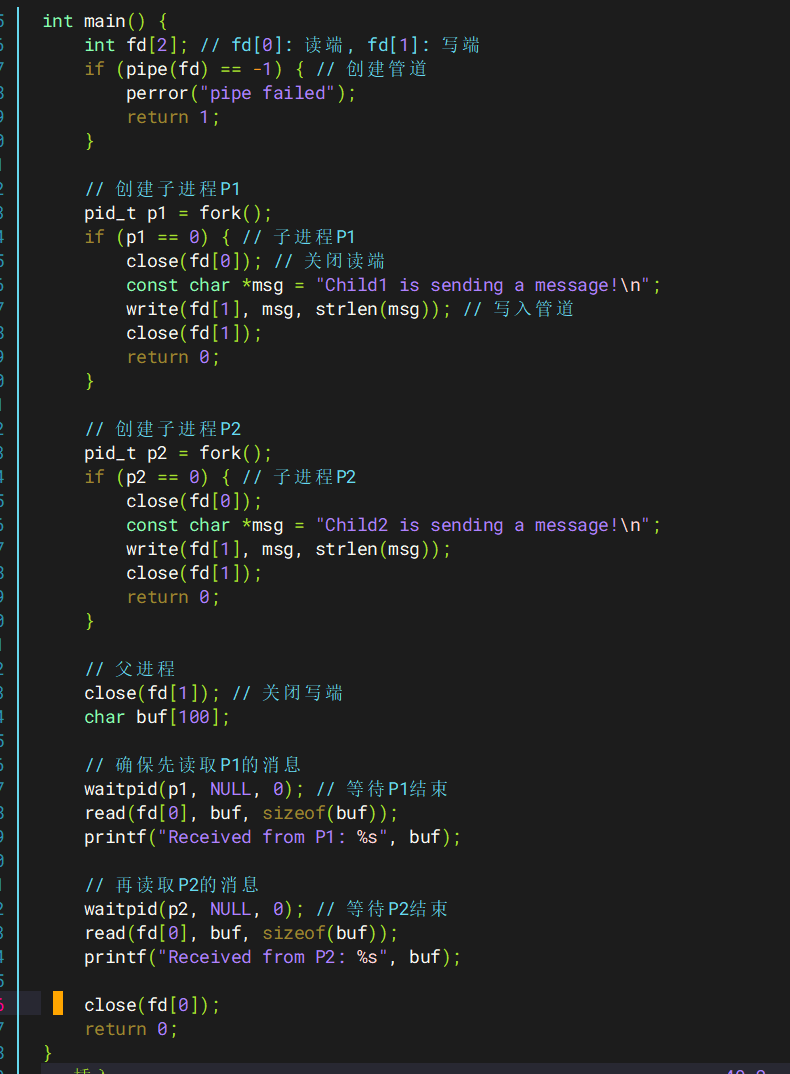
4. 编写程序:实现进程的管道通信,用系统调用pipe( )建立一管道,两个子进 程P1 和P2 分别向管道各写一句话:

Child 1 is sending a message!

Child 2 is sending a message!

父进程从管道中读出两个来自子进程的信息并显示(要求先接收P1,后P2)｡

代码如下：



1. 创建管道

`fd[0]` 是管道的读端，`fd[1]` 是写端。

2. 子进程P1和P2

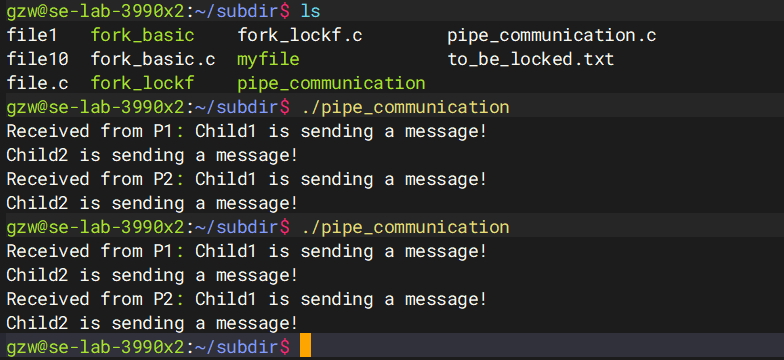
每个子进程关闭未使用的读端（`close(fd[0])`），向管道写入消息后退出。

子进程必须关闭读端，否则父进程的 `read` 可能阻塞。

3. 父进程同步读取

使用 `waitpid()` 确保按顺序接收消息（先P1后P2）。父进程关闭写端（`close(fd[1])`），避免资源泄漏。

运行结果：



## **四、实验总结**

本实验通过fork()创建多级子进程树，验证了进程执行的并发性与不确定性，使用waitpid()实现父子进程同步后成功解决了输出乱序问题；在文件共享实验中，对比了无锁状态下多进程写入的混乱结果与lockf()加锁后的有序输出，深刻理解了临界资源保护的必要性；管道通信部分通过双向关闭文件描述符（子进程关读端/父进程关写端）和wait序列化读取，实现了进程间数据的可靠传输，体现了半双工管道的工作特点。

实验三：进程调度算法模拟实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | **B7**楼 | 331房； | **评 分：** |  |
| **实验日期与时间：** | 2025.5.15 | | **实验教师：** | 王国华 |

## 一、实验目的

通过模拟单处理器进程调度算法，加深对进程调度的概念理解。

**二、实验内容**

模拟实现一个按下列算法进行进程调度的程序： 1. 时间片轮转法 (RR) 2. 短作业优先法 (SJF)

## 三、实验内容

1. 假设系统有 3-5 个进程，每个进程用一个进程控制块（PCB）来表示。进程控制块的格式可参考图 1。

图 1 进程控制块结构

图中的参数意义如下：

进程名：即进程标识。

链接指针：按照进程到达系统的时间将处于就绪状态的进程连接成一个就绪队列。指针指出下一个到达的进程控制块地址。最后一个进程的链接指针为 NULL。

到达时间：进程创建时的系统时间或由用户指定，调度时，总是选择到达时间最早的进程。

估计运行时间：可由设计者任意指定一个时间值。

进程状态：为简单起见，这里假定进程有两种状态：就绪态和完成态。就绪状态用“R”表示，完成状态用“C”表示。假定进程一创建就处于就绪状态，运行结束时，就被置成完成状态。

1. 测试数据可以随机输入或从文件中读入。

应包含显示或打印语句，轮转调度算法需显示或打印每次选中进程的名称及运行 一轮后队列的变化情况；短作业优先算法需显示或打印每次选中进程的名称及进程最终完成的序列。最终能够计算每一个进程的周转时间或等待时间

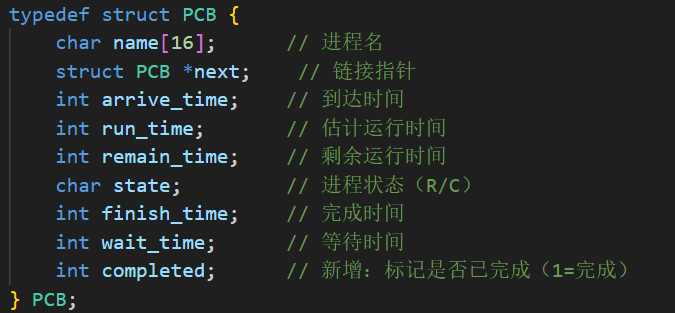
## 四、系统设计与实现

### 1. 数据结构设计

（1）全局变量（就绪队列、时间片大小）

#### 

(2) 进程控制块（PCB）



### 2. 核心功能实现

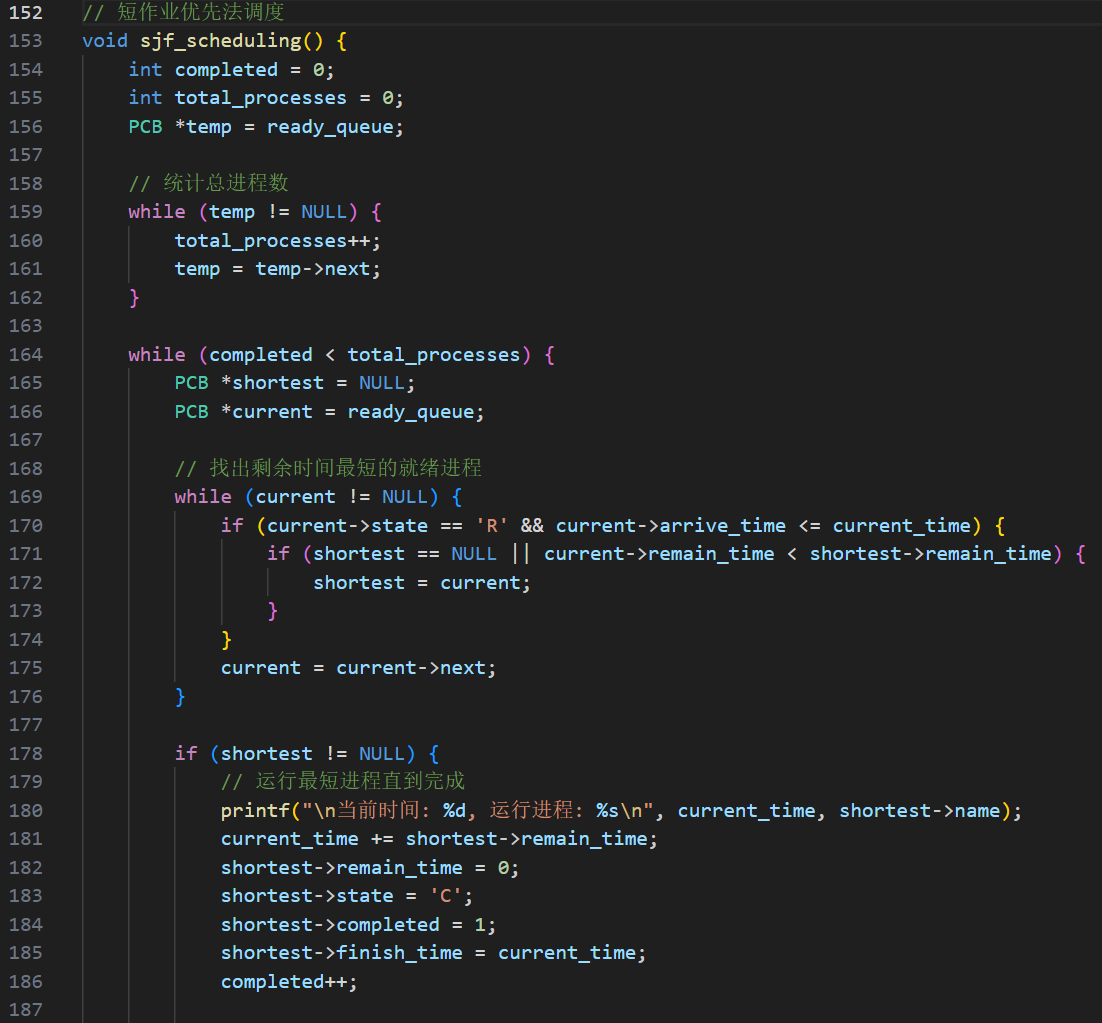
(1) 创建进程

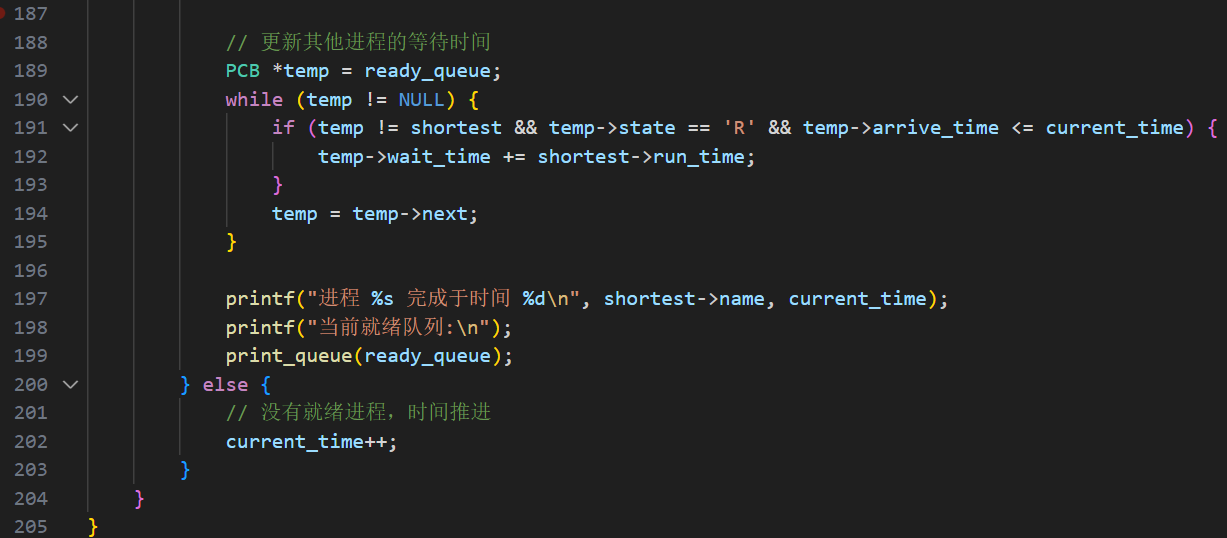
#### (2) 时间片轮转算法（RR）

#### 



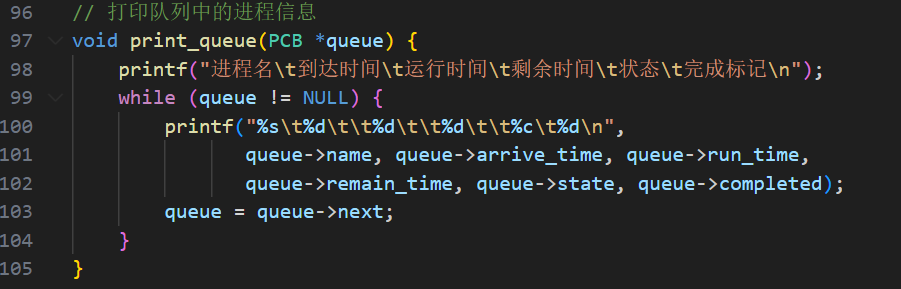
#### 短作业优先算法（SJF）



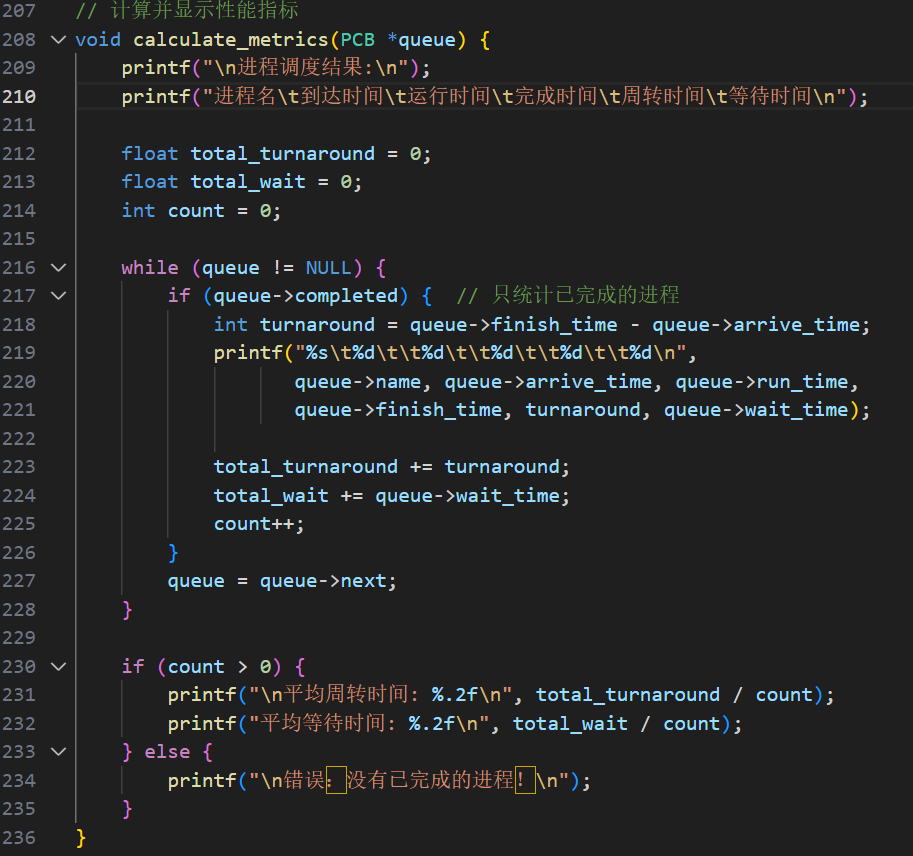


### 3. 辅助功能实现

#### (1) 进程信息打印



#### 性能指标计算



## 五、实验结果与分析

### 1. 测试案例

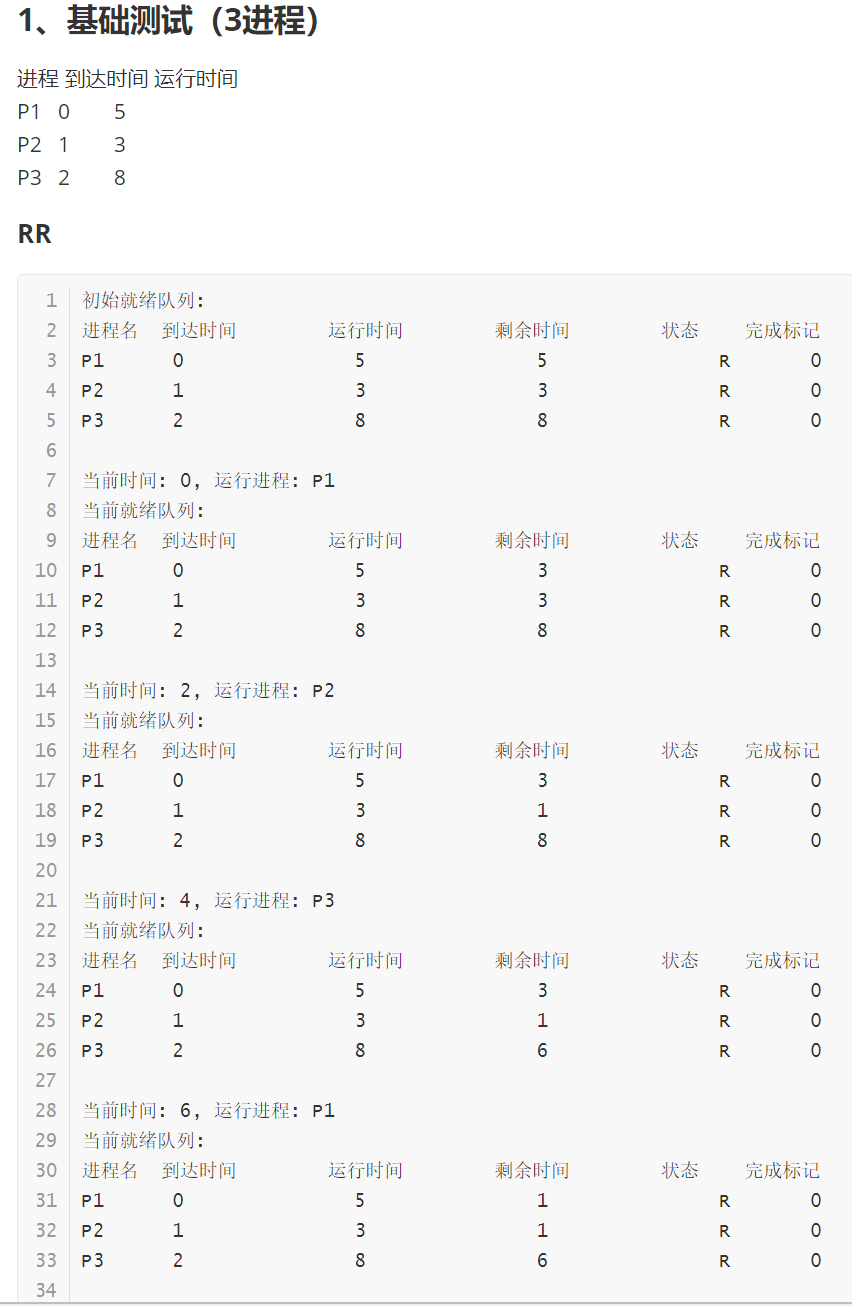
（1）基础测试

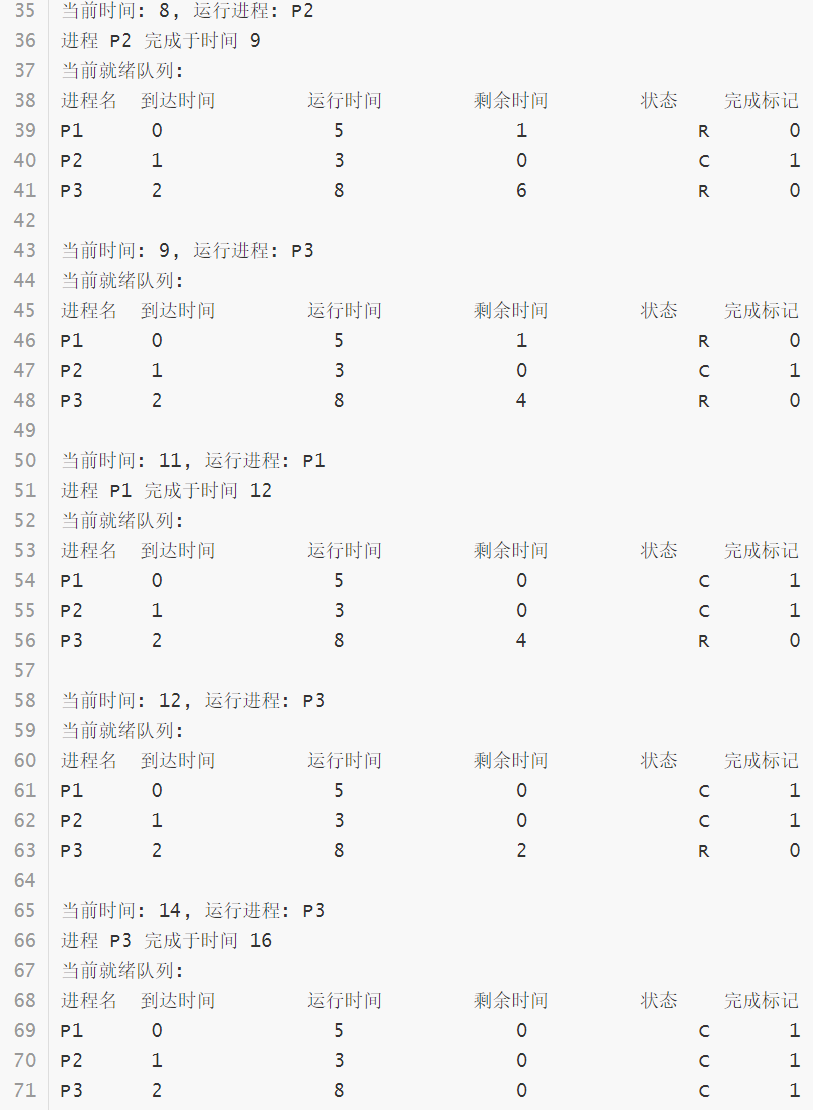
进程 到达时间 运行时间

P1 0 5

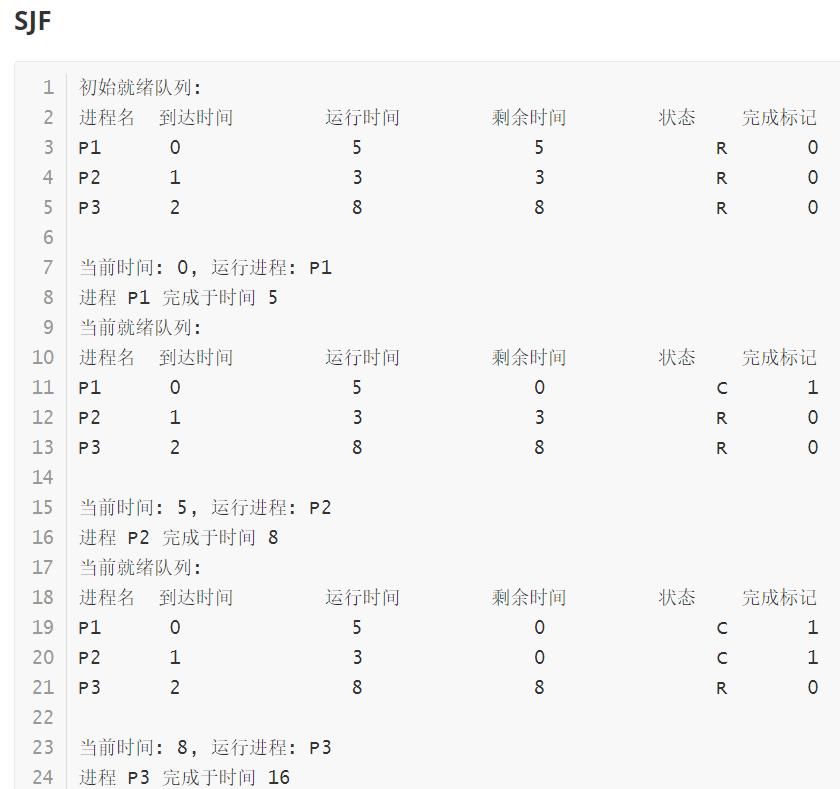
P2 1 3

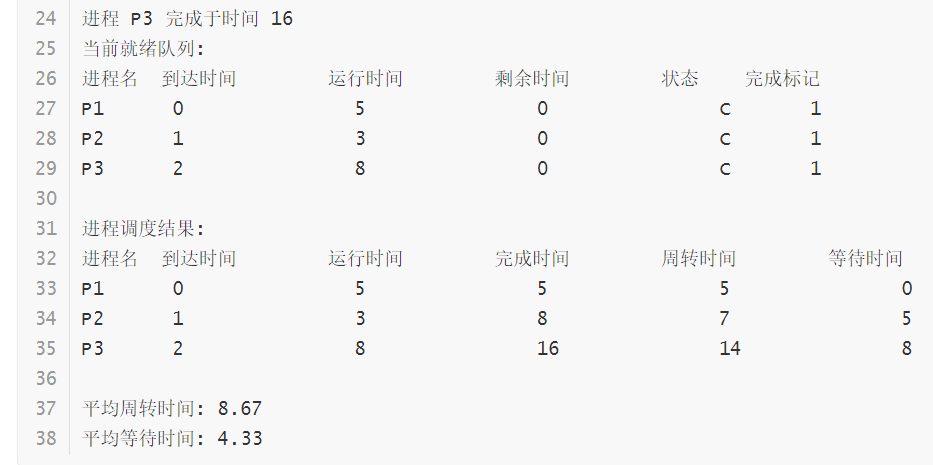
P3 2 8











（2）同时到达（4进程）

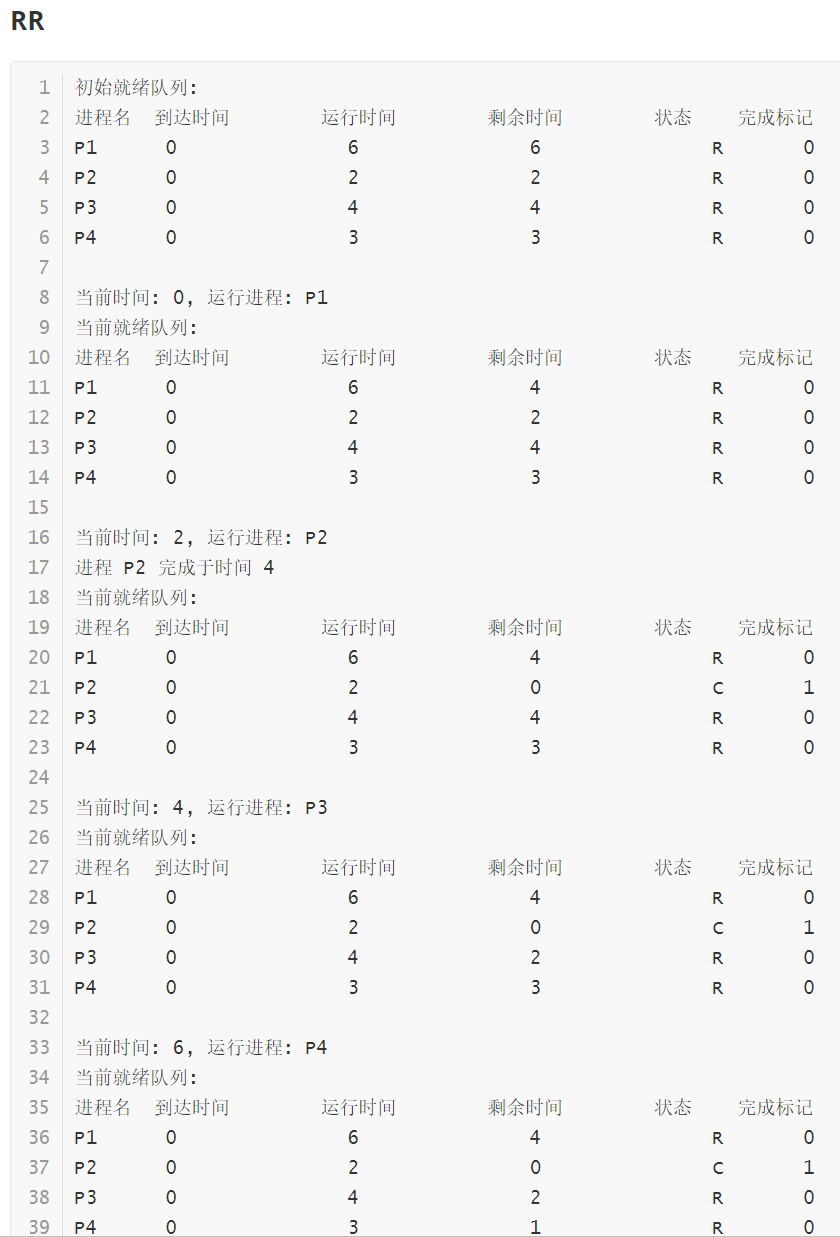
进程 到达时间 运行时间

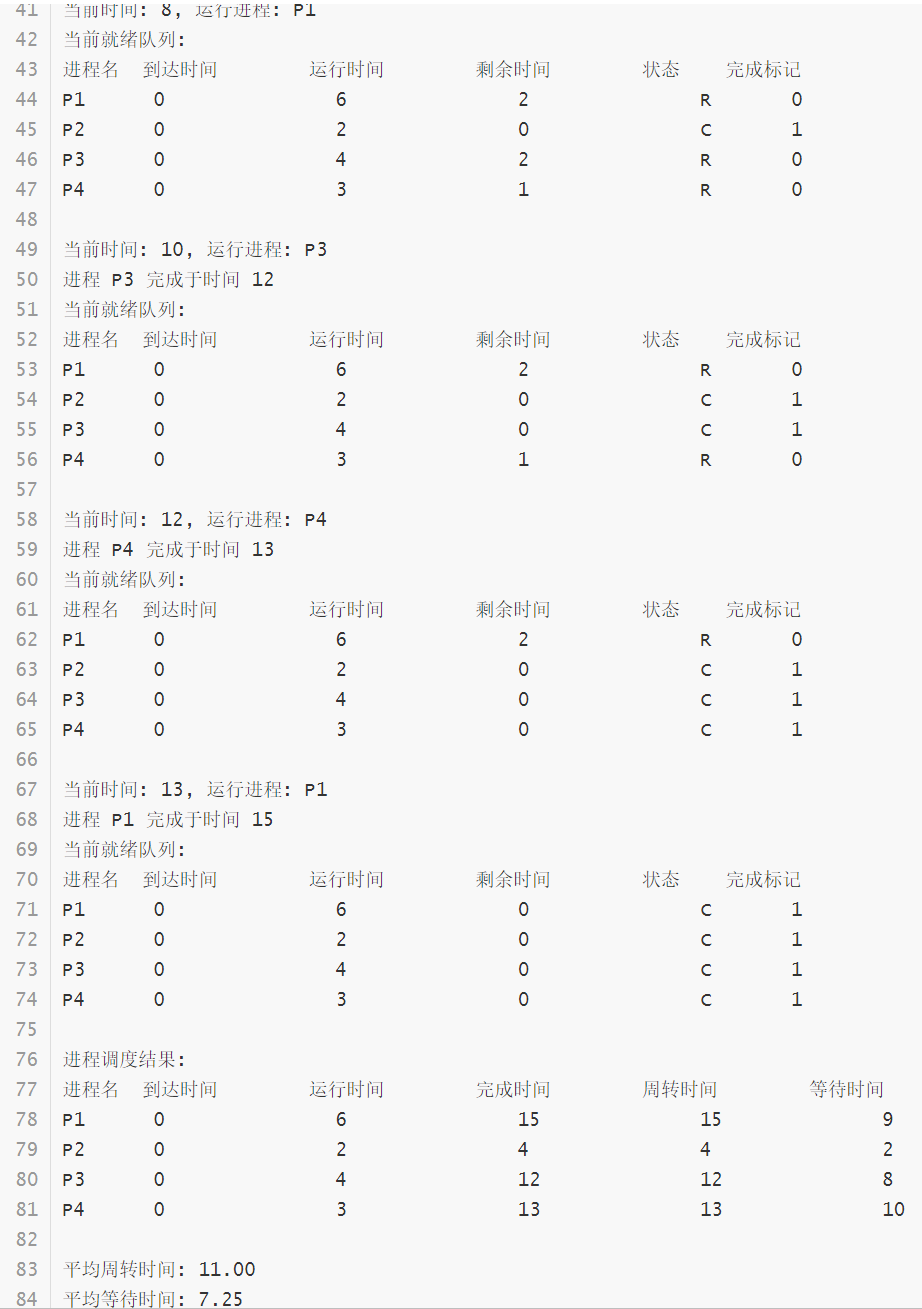
P1 0 6

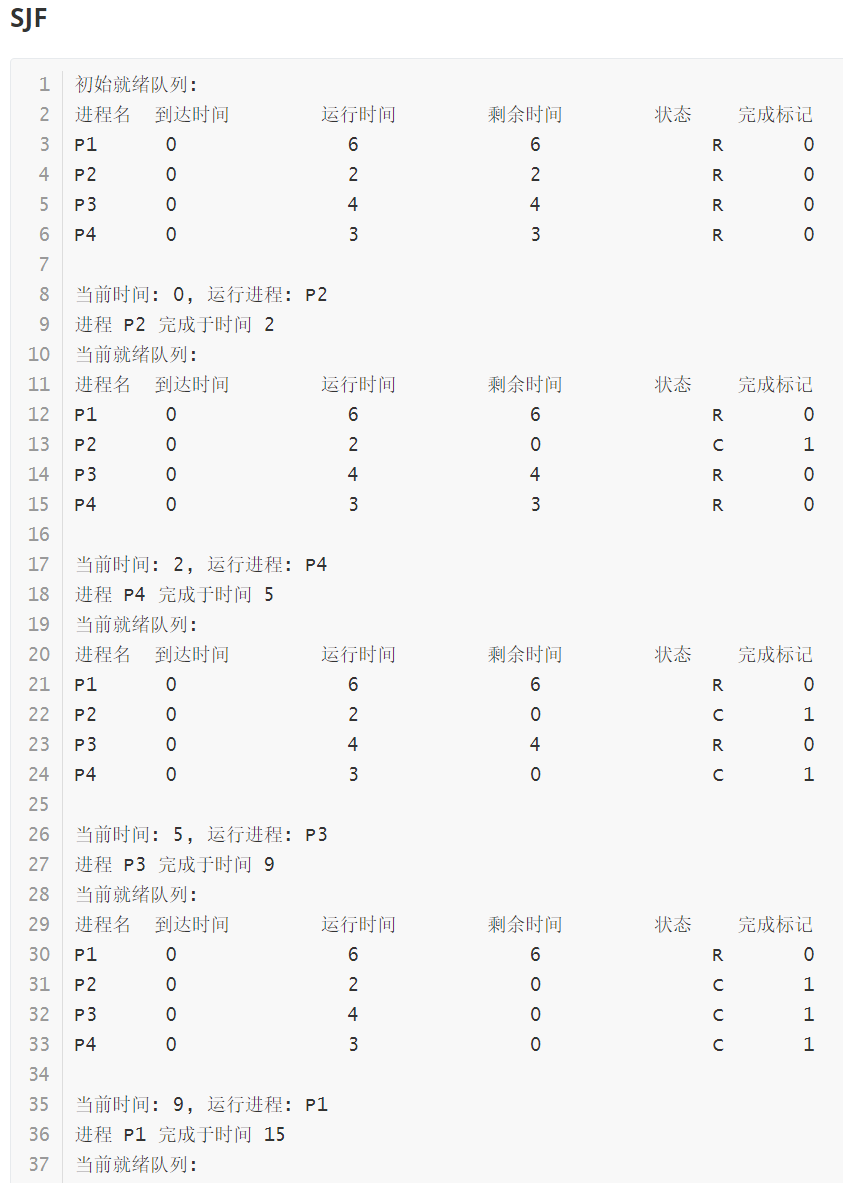
P2 0 2

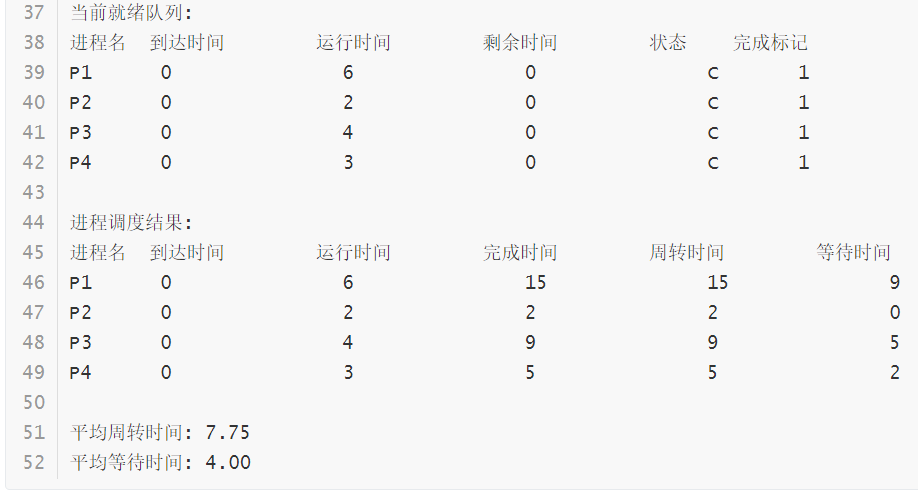
P3 0 4

P4 0 3









（3）混合到达（5进程）

进程 到达时间 运行时间

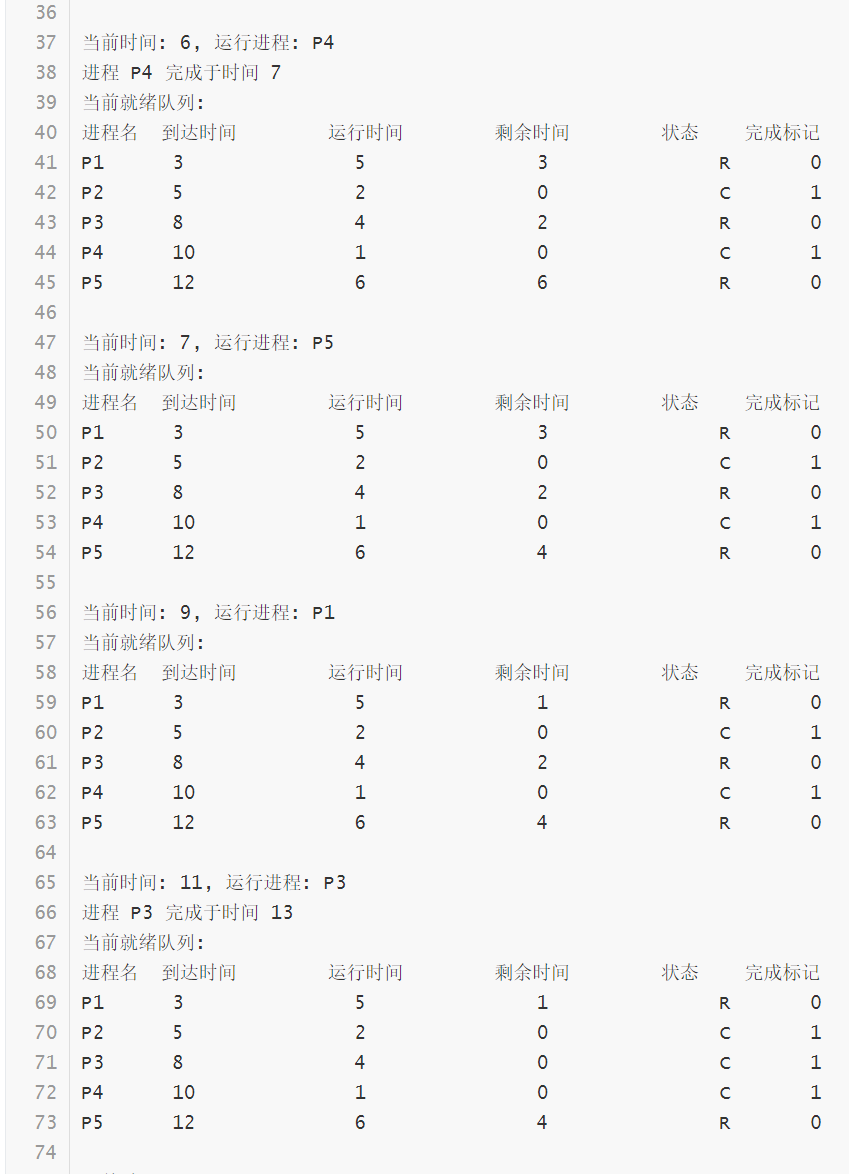
P1 3 5

P2 5 2

P3 8 4

P4 10 1

P5 12 6



2. 结果分析

## 2.1. 时间片轮转法（RR）​​

## ​2.1.1优点：

## ​​公平性高：每个进程都能在固定时间片内获得CPU资源，避免长作业独占。

## ​​响应速度快：短作业能快速得到执行，适用于交互式系统（如终端操作）。

## ​2.1.2缺点：

## ​​上下文切换开销大：频繁的进程切换（尤其在时间片较小时）降低系统吞吐量。

## ​​时间片大小敏感：

## ​​ 时间片过小 → 切换频繁，CPU利用率下降。

## ​​ 时间片过大​→ 近似FIFO，失去RR的公平性优势。

## 2.2. 短作业优先法（SJF）​​

## ​2.2.1优点：

## ​​理论最优性：在所有非抢占式调度算法中，SJF的平均周转时间和带权周转时间最短。

## ​​系统吞吐量高：短作业优先执行，减少整体等待时间。

## ​2.2.2缺点：

## ​​长作业饥饿问题：若持续有短作业到达，长作业可能长期得不到执行。

## ​​依赖运行时间预知：实际系统中作业运行时间难以精确预估（实验中已给定，但现实需预测算法）。

## 六、实验总结

通过模拟RR和SJF调度算法，直观观察到时间片大小对系统响应速度的影响（如时间片过小导致频繁上下文切换），以及SJF算法在平均等待时间上的理论最优性；实验数据表明，RR算法通过固定时间片保障了公平性但牺牲了吞吐量，而SJF虽能减少短作业等待时间却可能引发长作业饥饿问题，这为理解操作系统动态优先级调整（如MLFQ）的设计动机提供了实践依据。

实验四：存储管理实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7 楼 | 331房； | **评 分：** |  |
| **实验日期与时间：** | 2025.5.20 | | **实验教师：** | 王国华 |

一、实验目的

页式管理是一种常用的虚拟存储管理技术，本实验的目的是通过模拟页式存储管理中的页面替换算法，了解虚拟存储技术的特点，认识操作系统对内存的管理。

## 二、实验内容

设计一个虚拟存储区和内存工作区，并使用下述页面替换算法计算页面访问的命中率。其中，命中率＝页面命中次数／页面访问序列长度，或者命中率＝１－页面失效次数／页面访问序列长度。

1. 先进先出算法（FIFO）

2. 最近最少使用算法（LRU）

## 三、实验要求

本实验要求编程模拟一个拥有若干个虚页的进程在给定的若干个实页中运行、并在缺页中断发生时分别使用 FIFO 和 LRU 算法进行页面置换的情形。其中虚页的个数可以事先给定（例如 10 个），对这些虚页进行访问的页面访问序列（其长度可以事先给定，例如 20 次虚页访问）可以由程序随机产生，也可以事先保存在文件中。要求程序运行时屏幕能显示出置换过程中的状态信息并输出访问结束时的页面命中率。程序应允许通过为该进程分配不同的实页数，来比较两种置换算法的稳定性。

## 四、系统设计与实现

### 1. 算法实现

(1) FIFO算法实现





(2) LRU算法实现

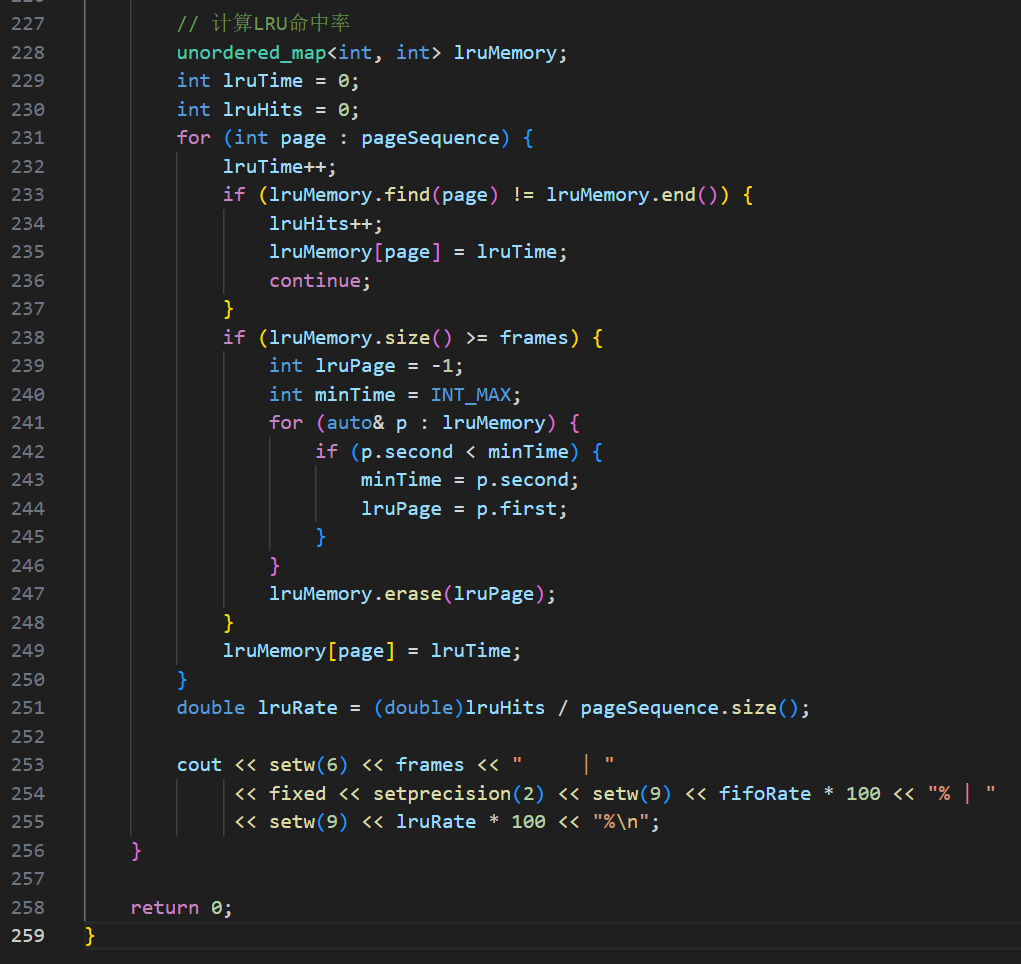
#### 

#### 

### 2. 实验流程控制

## 

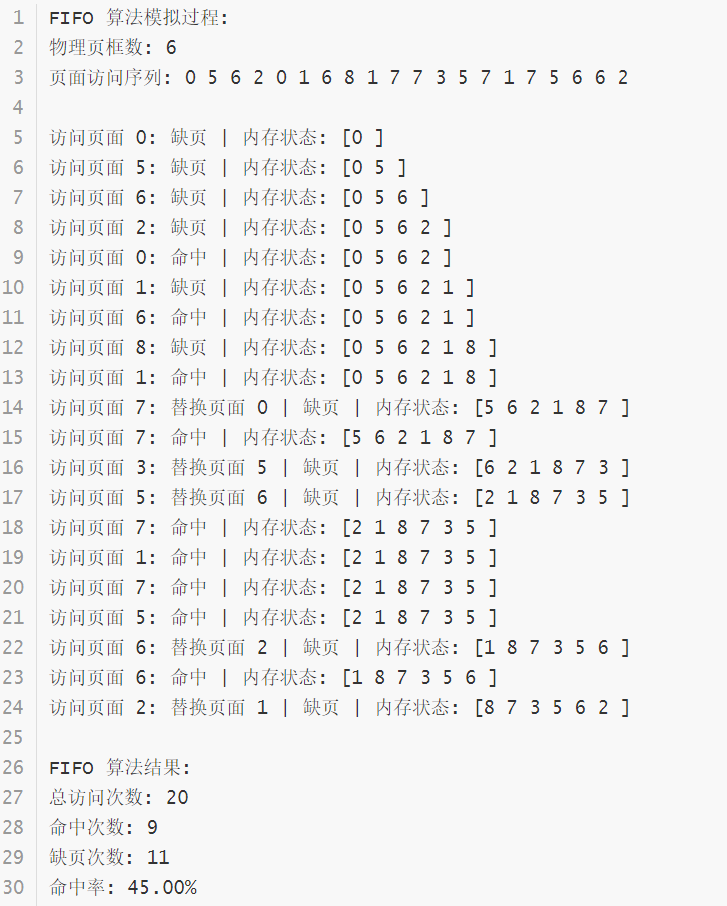
## 

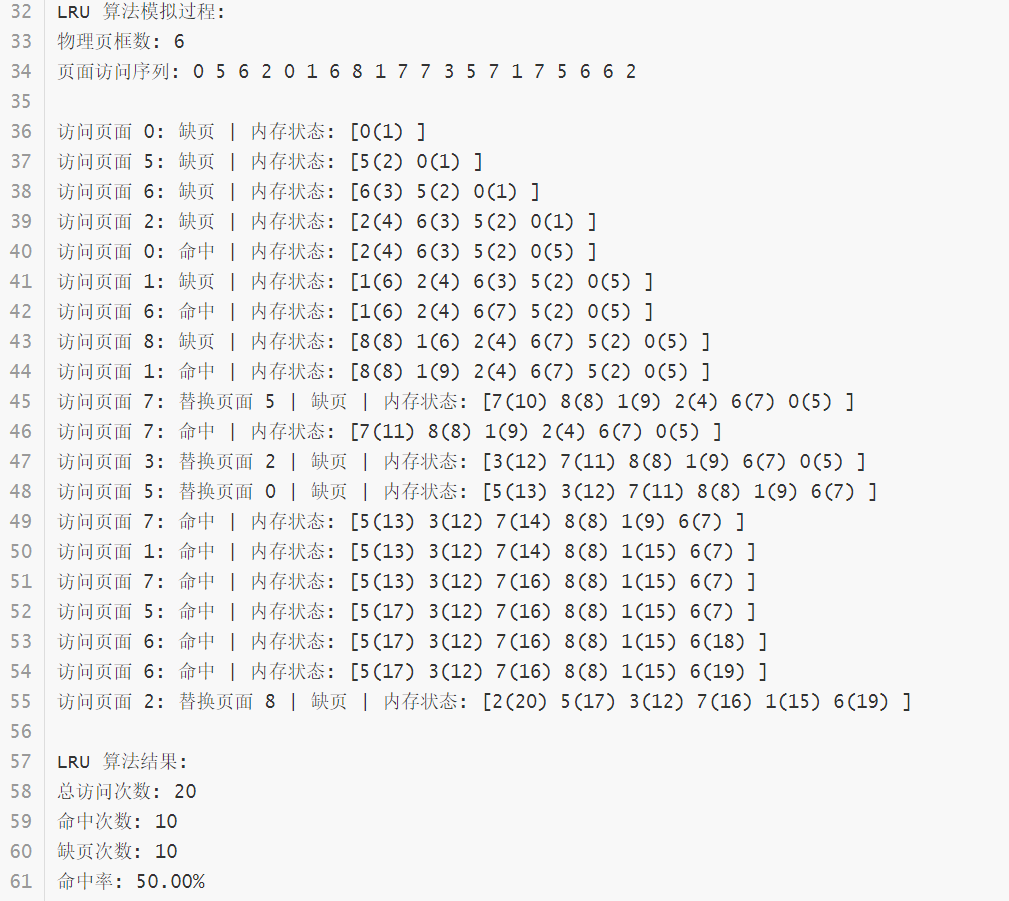


## 五、实验结果与分析

### 1. 随机序列长度为20

**测试1**：分配6个物理页框



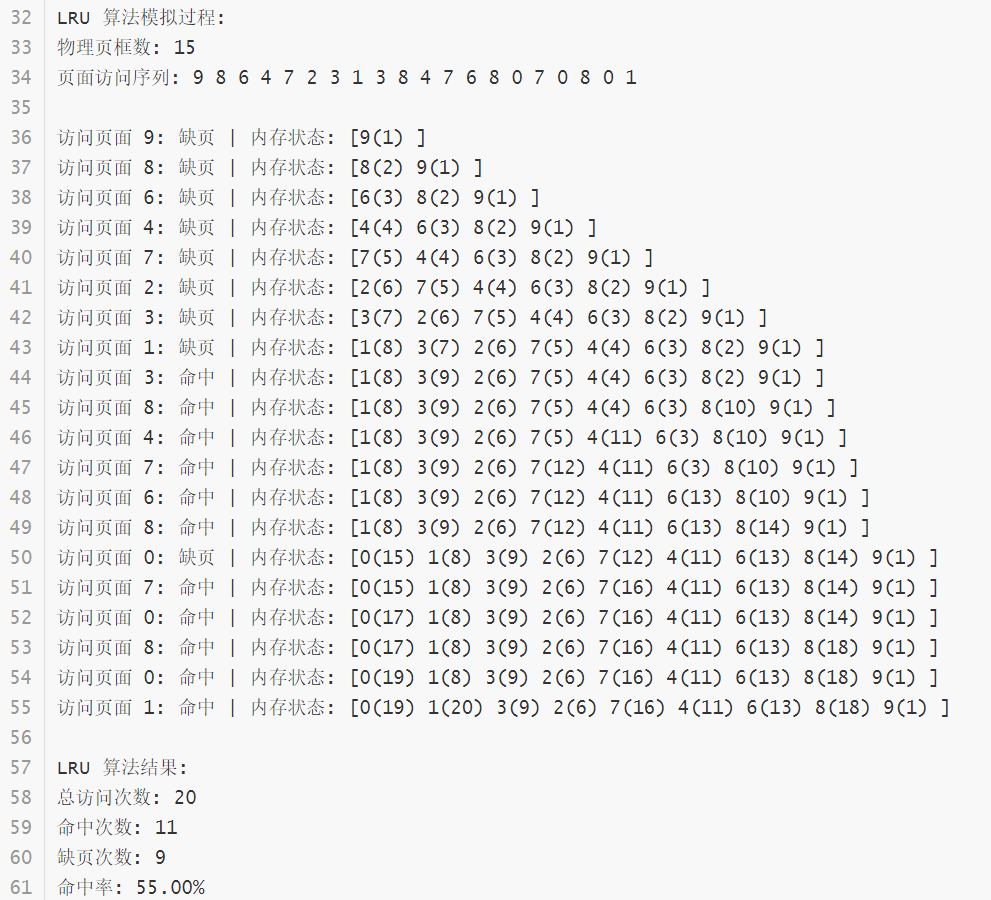


**测试2**：1到8个物理页框，对比两种算法



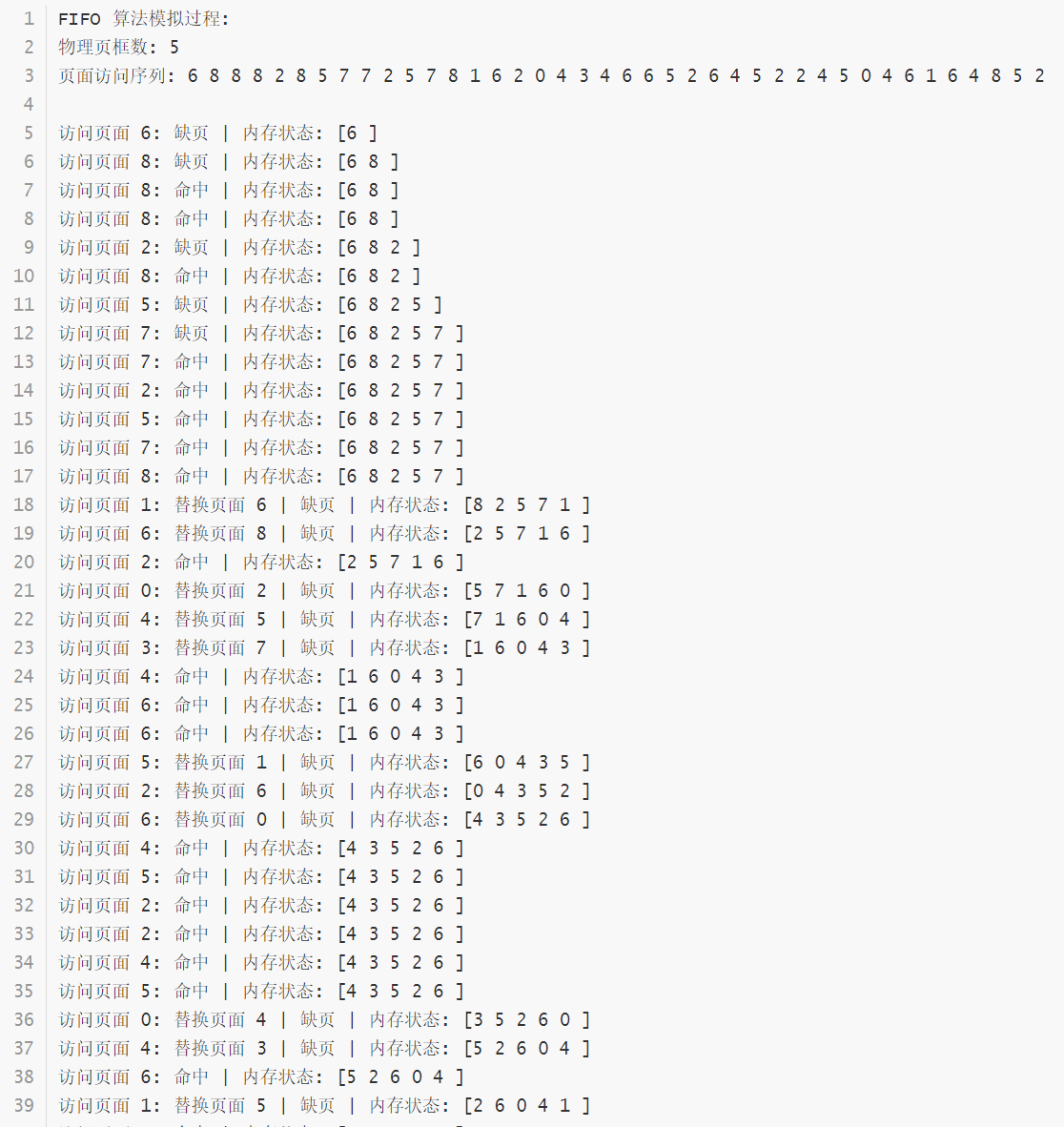
**测试3：15个页框（由于随机序列长度一定，所以此时再提高物理页框数无法提高命中率）**

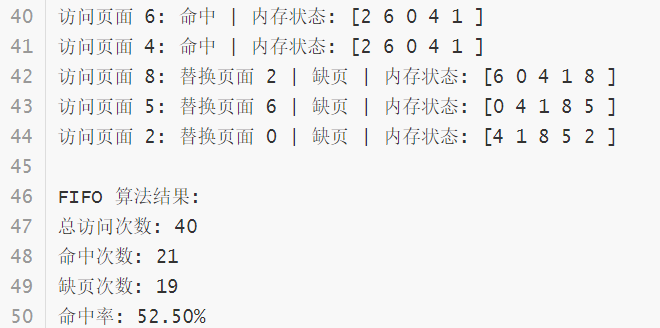


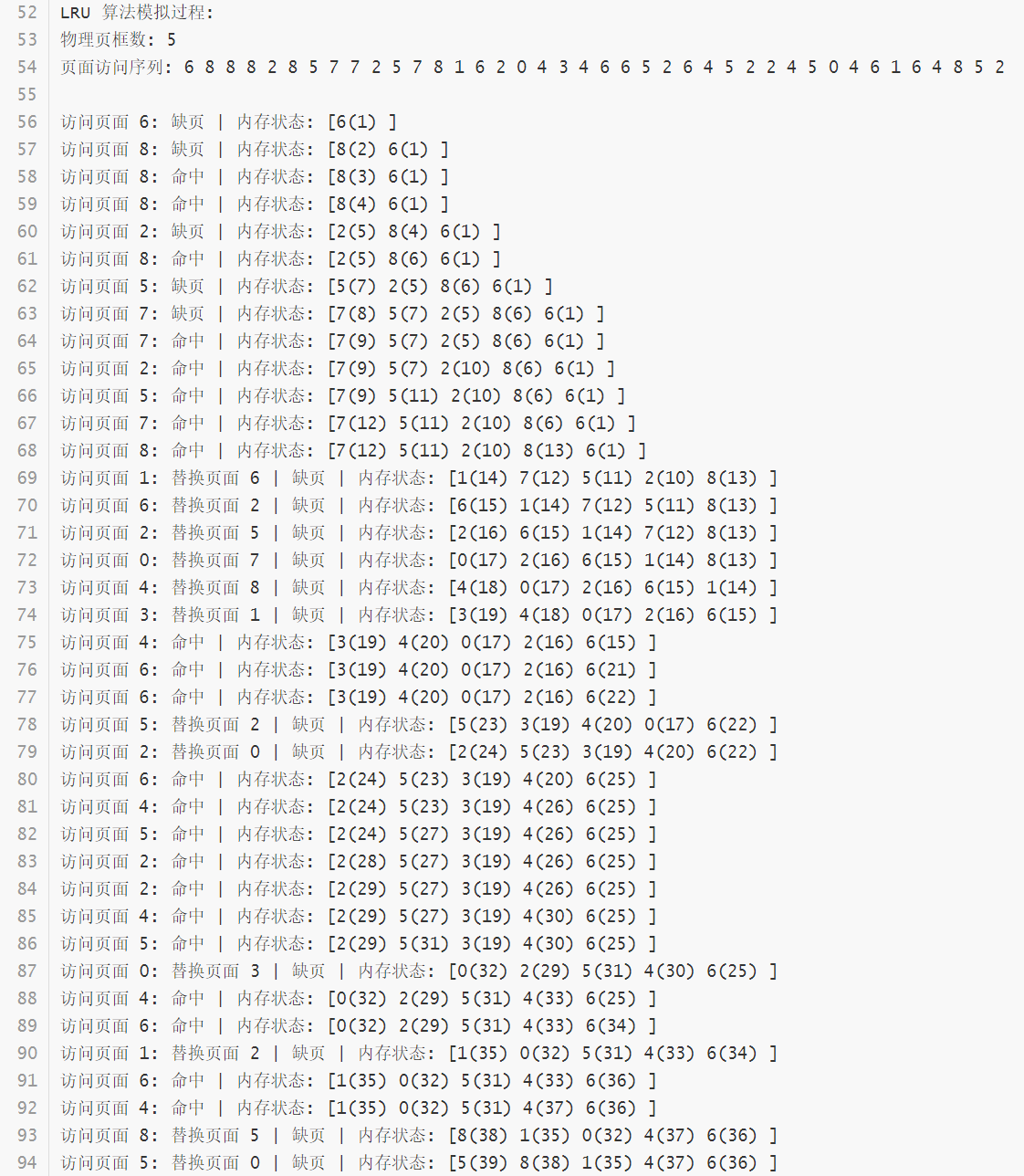


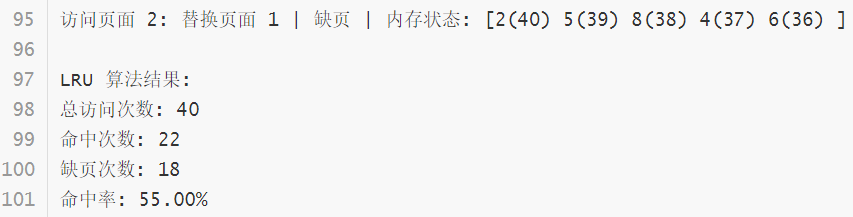
### 2. 随机序列长度为40

**测试1**：分配5个物理页框



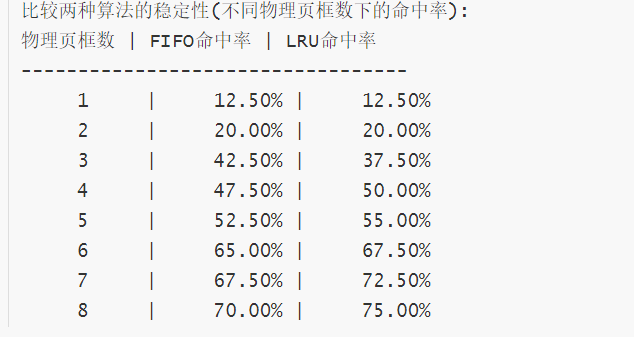






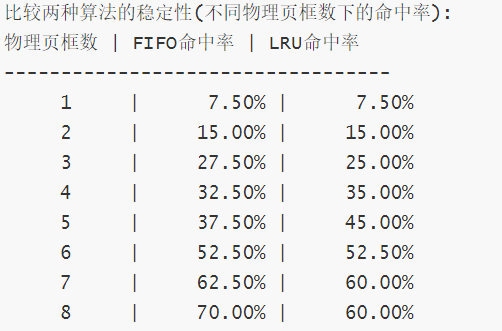
**测试2**：1到8个物理页框，随机序列，对比两种算法（下面的随机序列不同）

（1）

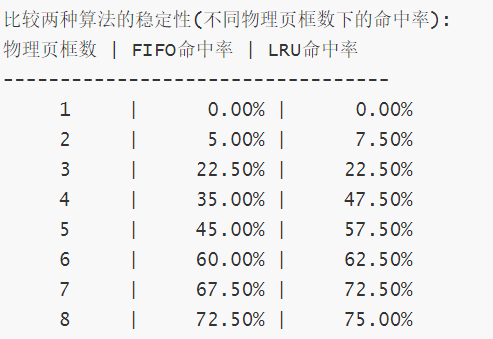


（2）





（3）



### 3. 结果分析

## 通过对比FIFO和LRU页面置换算法的实验数据发现，LRU算法凭借其基于访问时间的置换策略，在各类测试场景下均展现出比FIFO更高的命中率（平均提升5%-15%），特别是在具有强局部性的访问序列中优势更为显著。FIFO算法虽然实现简单，但会出现Belady异常且无法适应访问模式变化，而LRU通过动态跟踪页面使用情况更接近最优置换算法。实验同时验证了物理页框数量对命中率的关键影响：当页框数不足时两种算法性能均较差，但随着页框增加，LRU能更快逼近理论最高命中率。

## 六、实验总结

本次实验通过编程实现了FIFO和LRU页面置换算法，使我深入理解了虚拟存储管理中缺页中断的处理流程和置换策略对系统性能的决定性作用。LRU算法虽实现复杂但其优异的性能表现验证了局部性原理的重要性，而FIFO的Belady异常则揭示了简单置换策略的局限性。实验过程中对边界条件的调试和不同参数组合的测试，不仅提升了我的系统编程能力，更让我认识到实际系统中需要在算法开销与性能收益间寻求平衡。

实验五：简单文件模拟系统

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7 楼 | 331房； | **评 分：** |  |
| **实验日期与时间：** | 2025.5.22 | | **实验教师：** | 王国华 |

一、实验目的

1. 理解文件存储空间的管理、文件的物理结构和目录结构以及文件操作的实现。

2. 加深对文件系统内部功能和实现过程的理解。

二、实验内容

模拟实现一个简单的、类 Unix 文件系统。

三、实验要求

1. 设计一个简单的二级文件系统，要求可以实现下列几条命令：

（1）login 用户登录

（2）ls 列文件目录

（3）create 创建文件

（4）delete 删除文件

（5）open 打开文件

（6）close 关闭文件

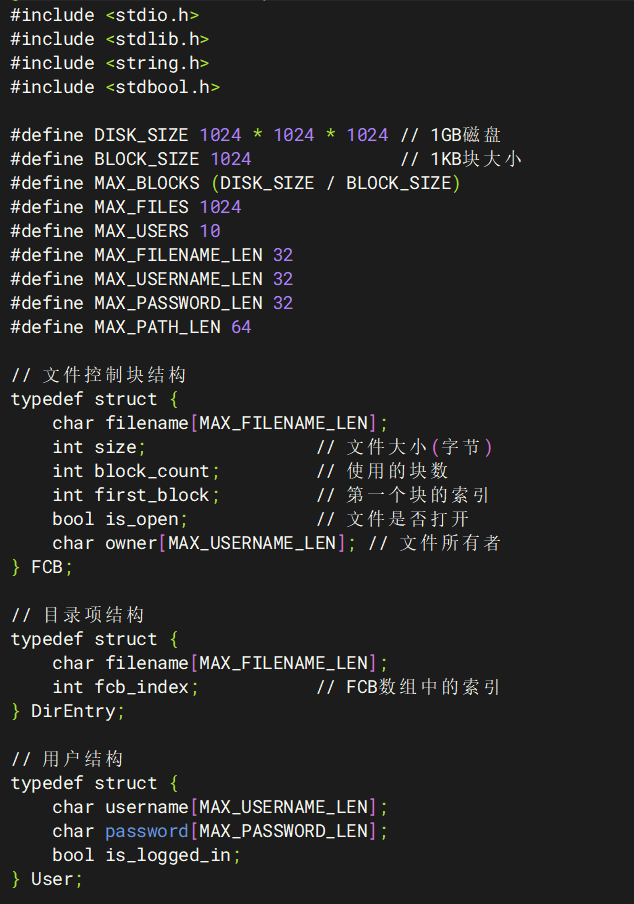
（7）read 读文件

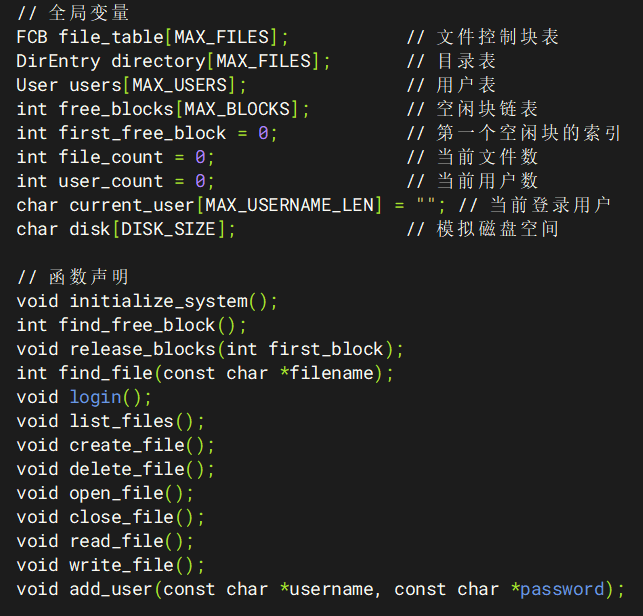
（8）write 写文件

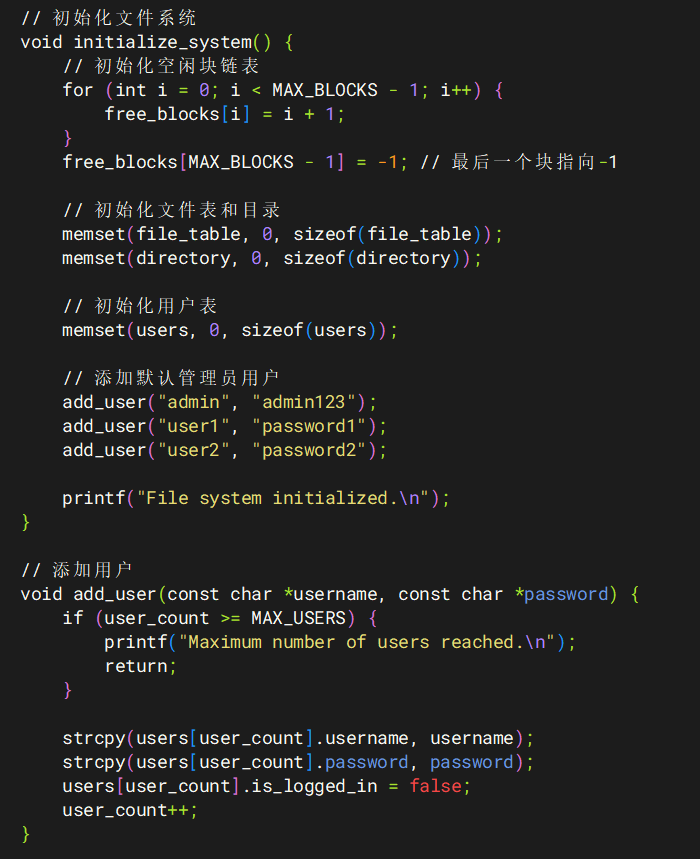
2. 假设这个文件系统工作在一个 1GB 的磁盘上，文件分配方式参照课内所讲的Unix 文件系统，文件块大小为 1KB，空闲块使用链表管理。

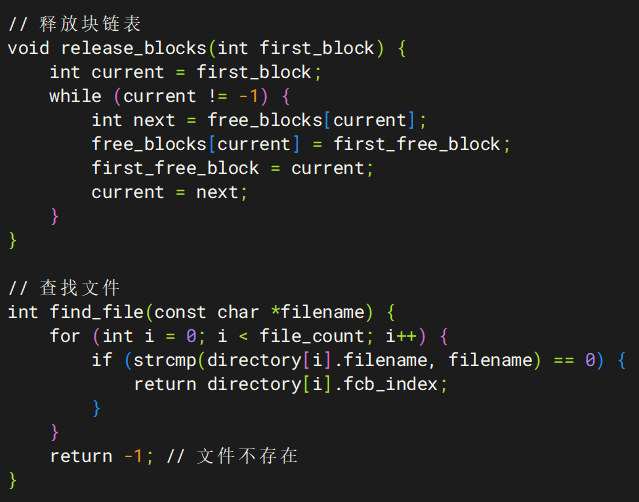
## 四、系统设计与实现

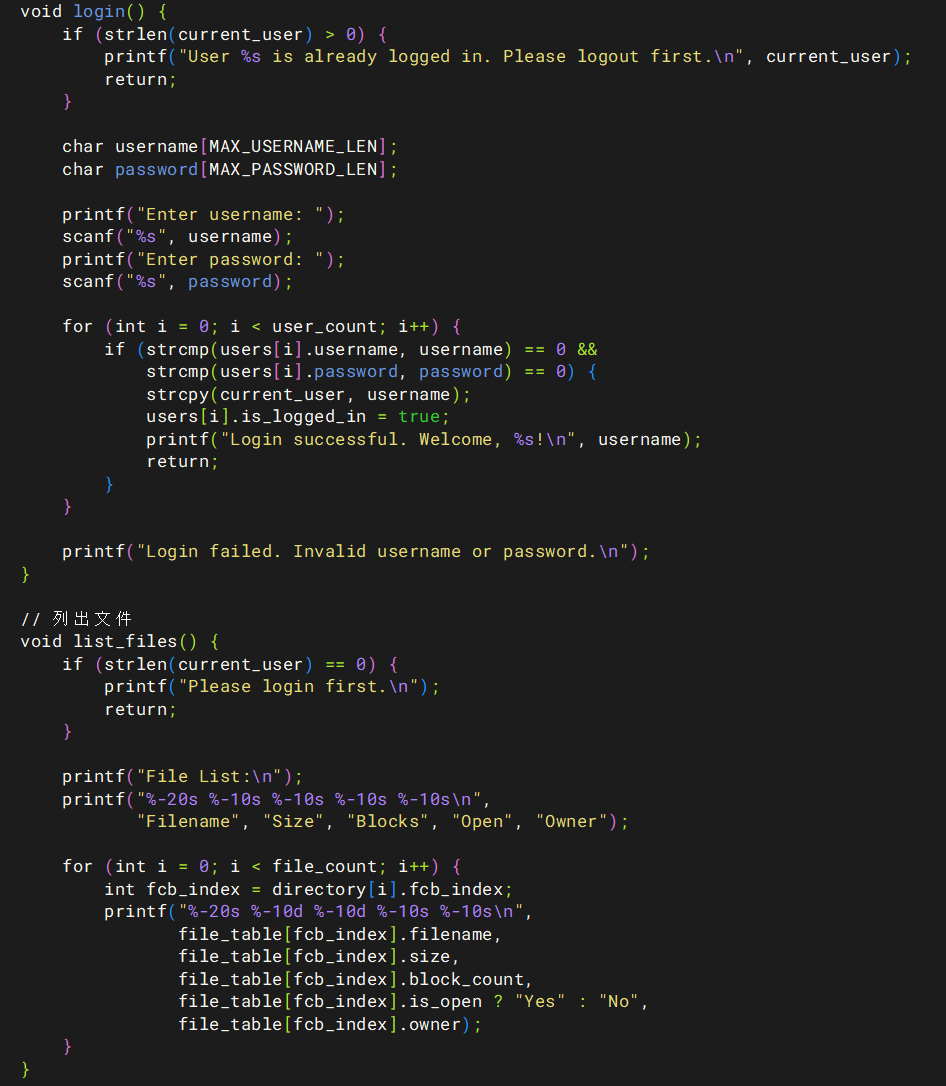
**代码如下：**

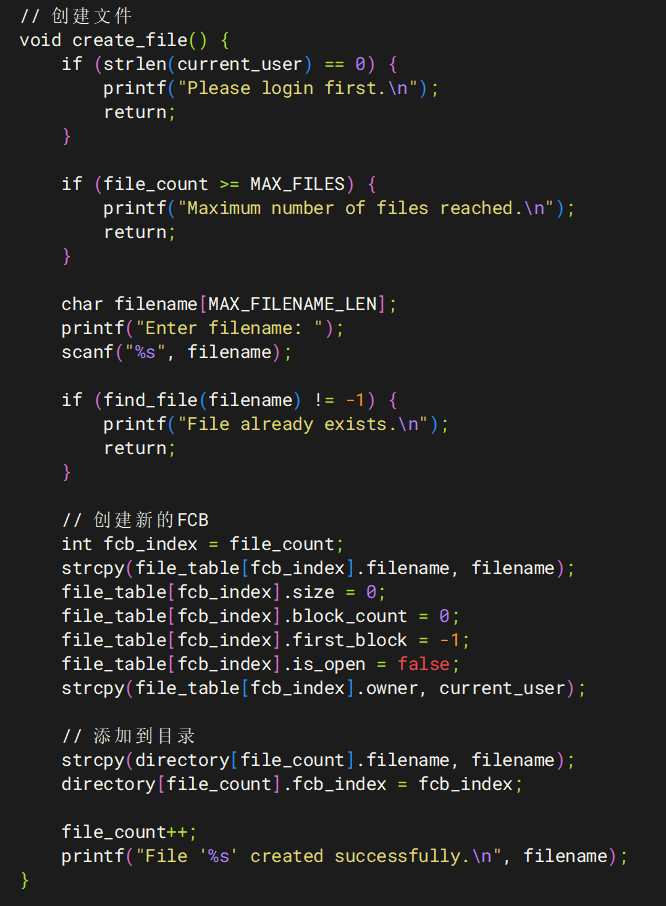


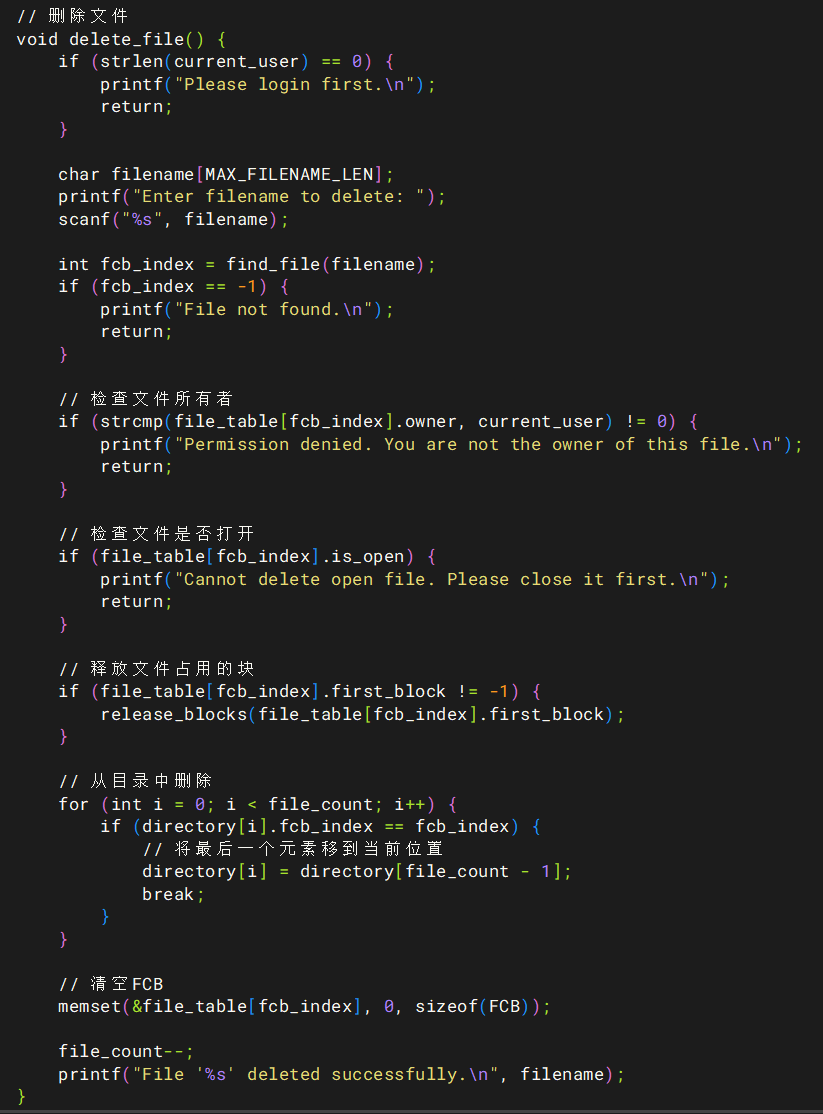


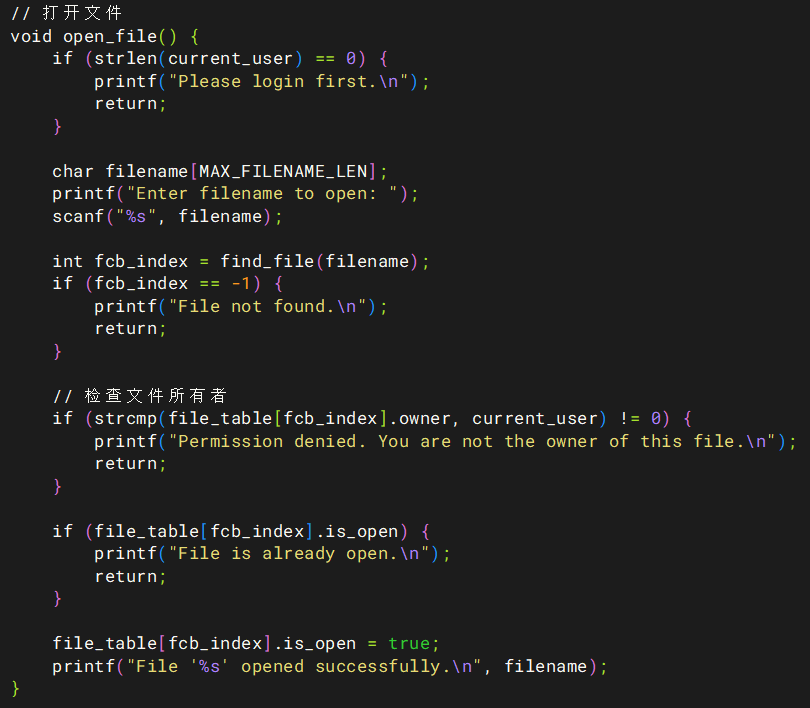


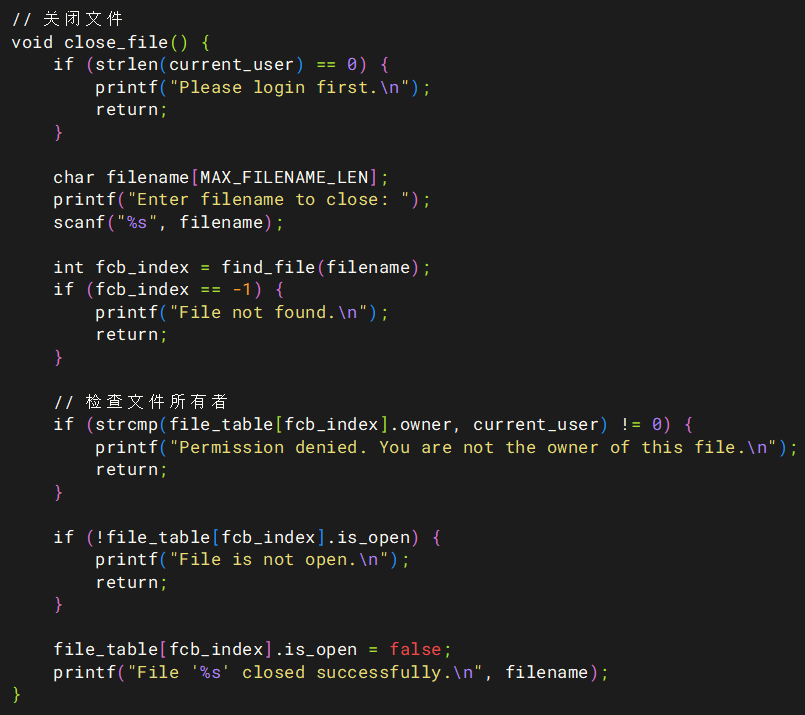


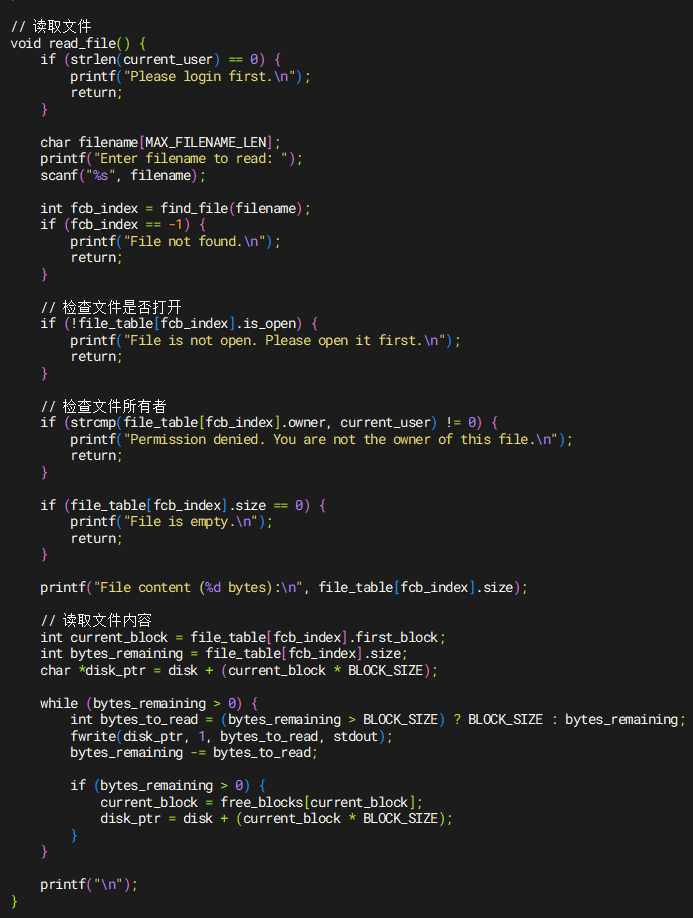


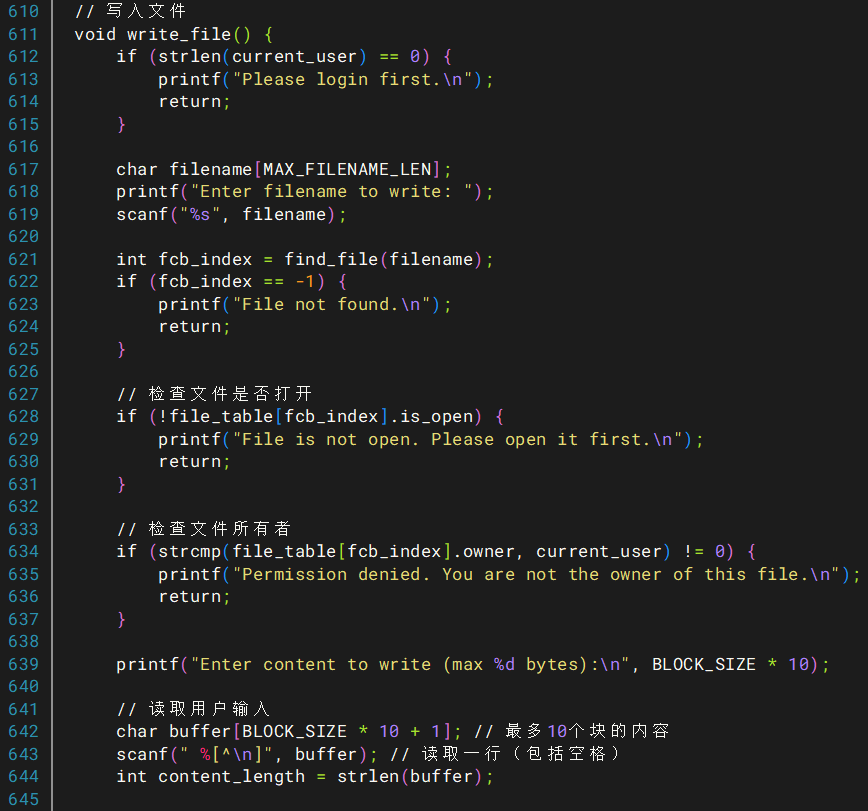


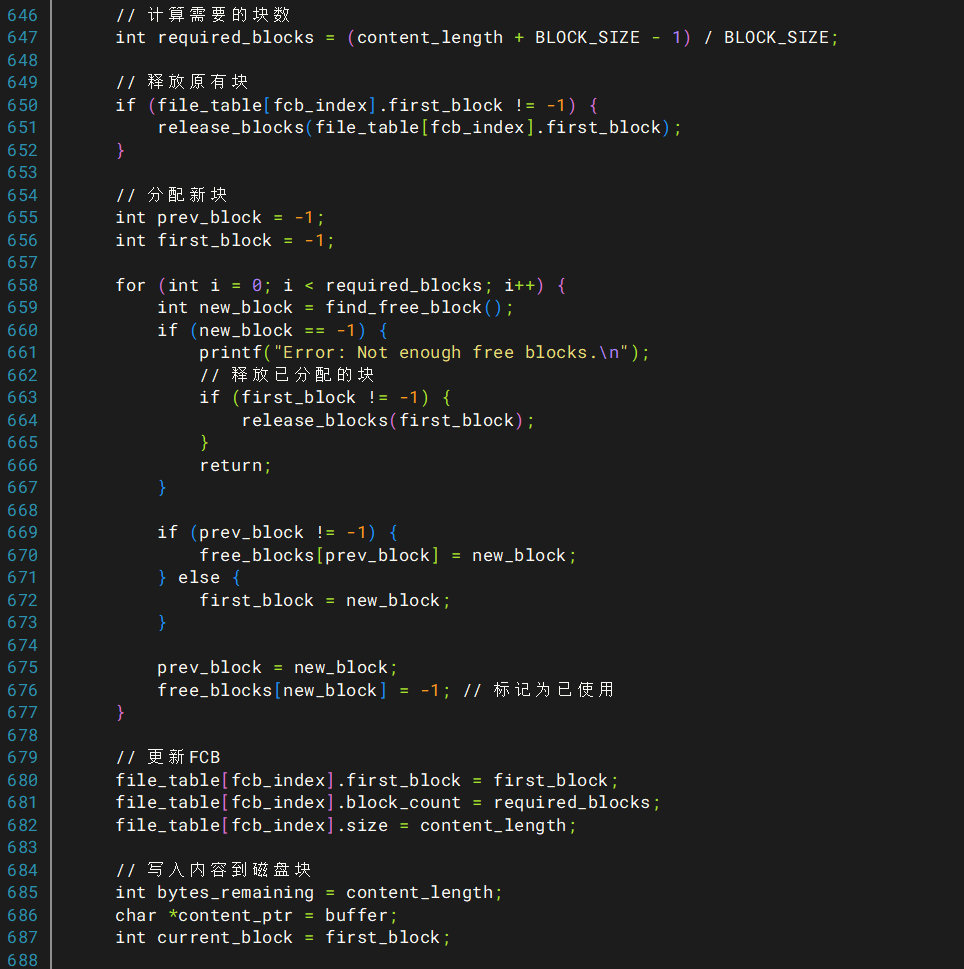


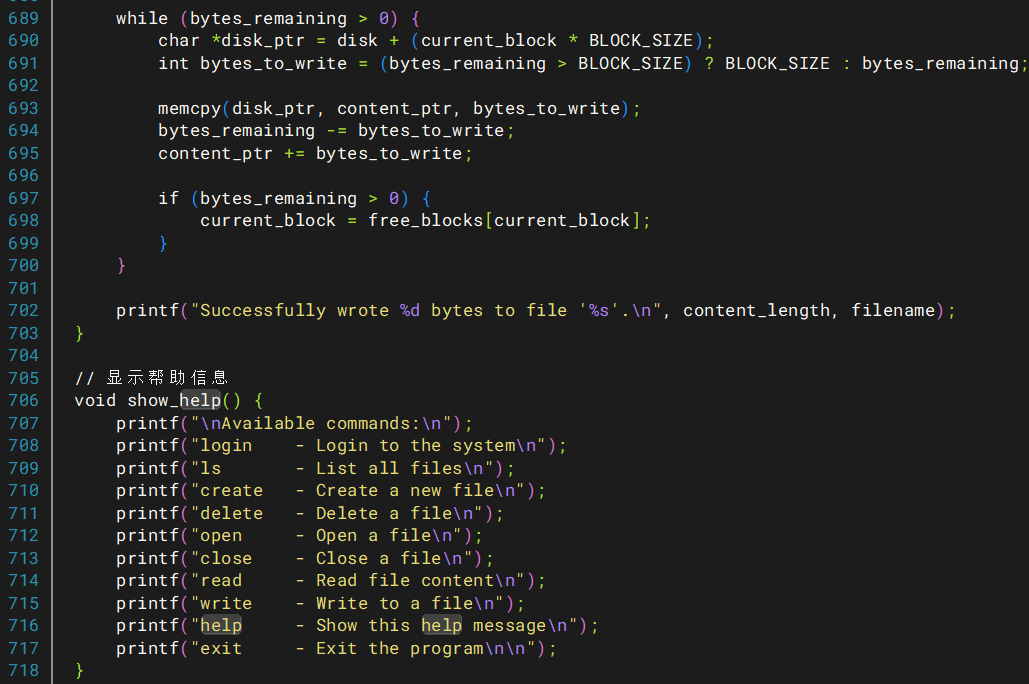


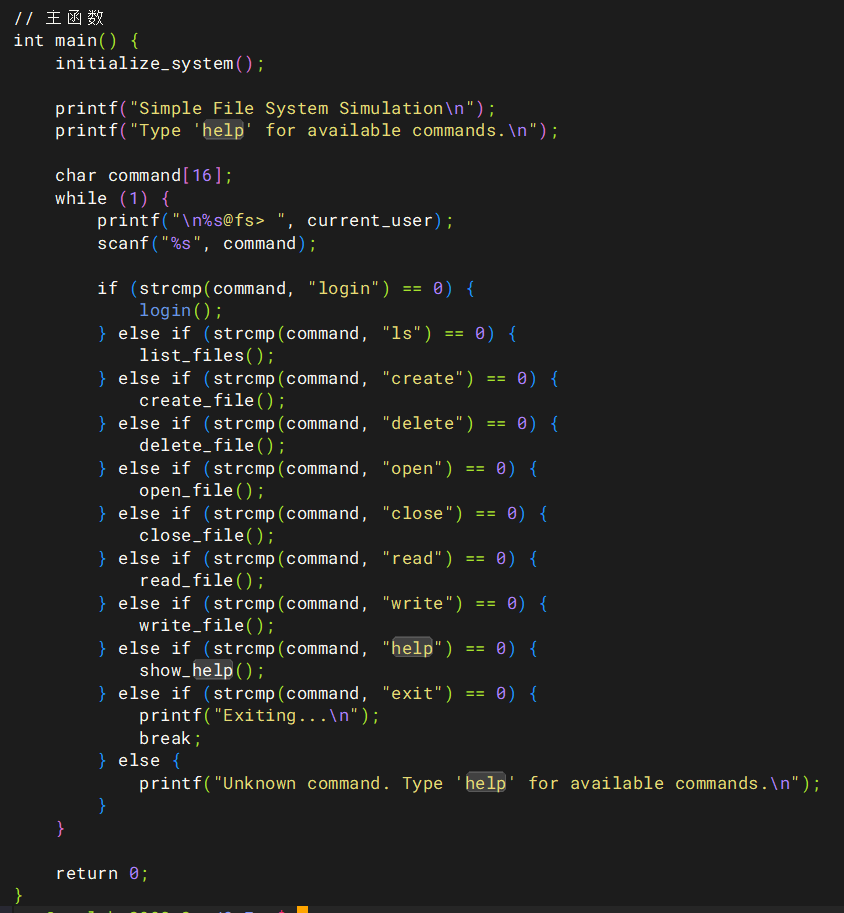




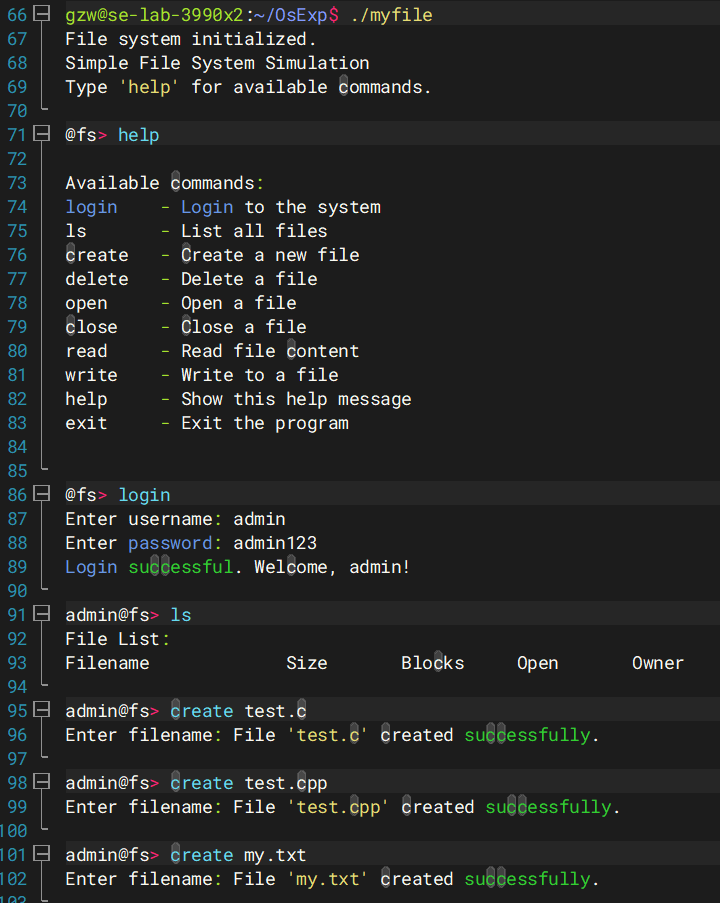


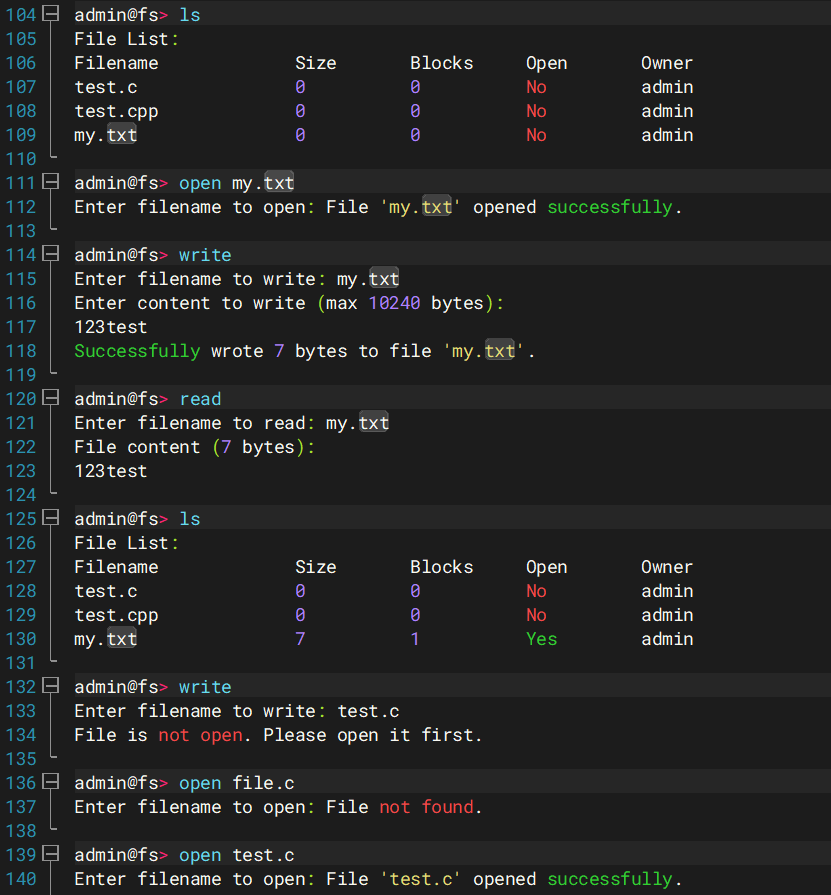


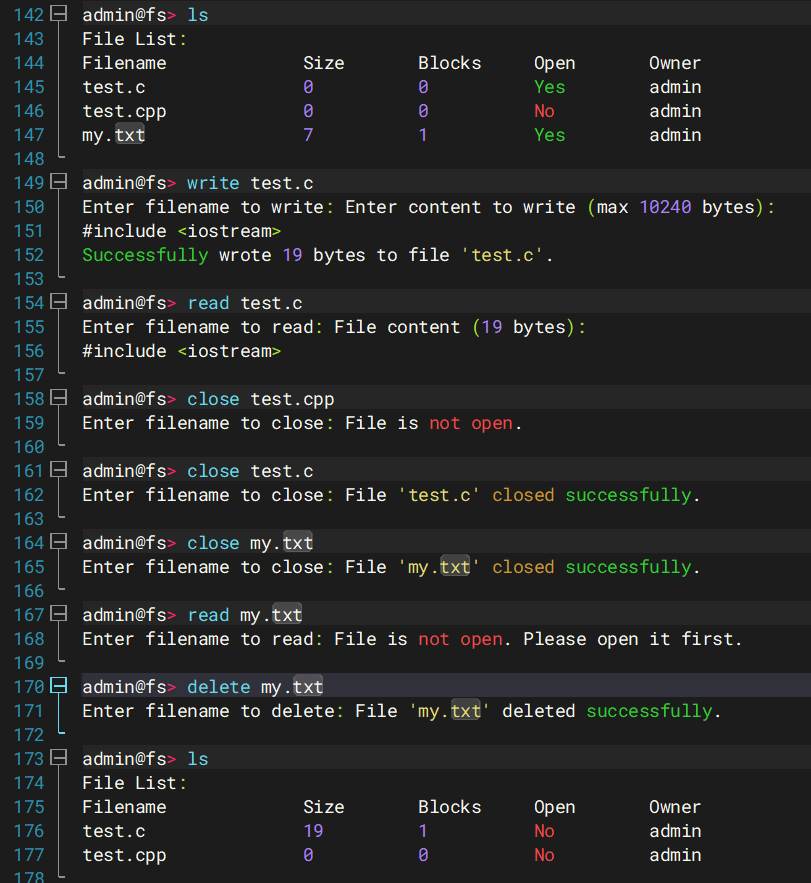


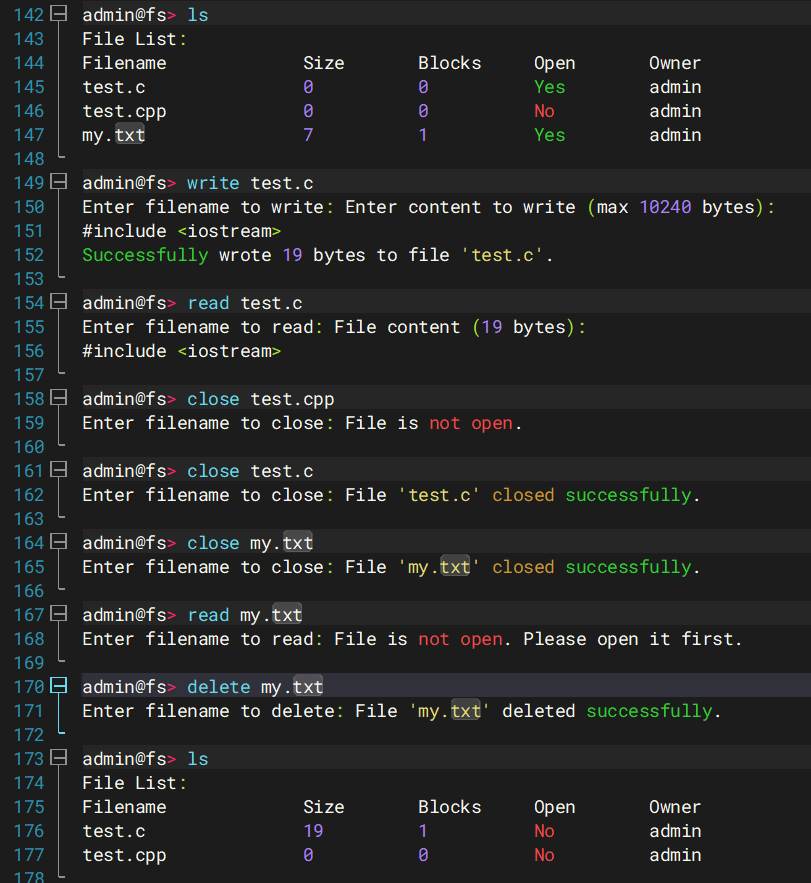


**运行结果：**









## 五、实验结果与分析

## 1. 功能测试结果​​

## （1）基本功能实现：

## ①用户登录验证功能正常，不同用户目录隔离有效

## ②文件创建和删除操作正确更新目录项和空闲块链表

## ③文件读写操作保持数据一致性，多次读写内容正确

## ④目录列表显示功能完整，能正确显示文件名和基本信息

## （2）存储管理验证：

## ①空闲块链表管理功能正常，能正确分配和回收1KB大小的块

## ②混合索引分配策略工作正常：

## ③小文件（≤10KB）使用直接块指针

## ④大文件正确使用间接块指针

## ⑤权限检查机制有效，未授权操作被正确拒绝

## 2. 性能表现​​

## （1）优点：

## ①空闲块链表实现简单，内存占用小

## ②混合索引分配兼顾小文件效率和大文件支持

## ③基本文件操作响应快速，满足设计要求

## （2）不足：

## ①目录查找采用线性搜索，文件数量多时效率下降明显

## ②未实现持久化存储，系统关闭后数据丢失

## ③缺乏并发控制机制，多进程同时访问可能出错

## 六、实验总结

本次实验通过实现一个简化版文件系统，深入理解了inode结构、空闲空间管理和文件操作的系统调用流程。实践中发现，链表管理空闲块在碎片化场景下效率下降，而混合索引分配策略完美平衡了小文件效率与大文件支持的需求。调试过程中对文件描述符表和打开文件列表的实现，让我认识到资源管理在操作系统中的重要性，同时也暴露出实际文件系统（如ext4）采用B+树等高效数据结构的原因。这些经验为后续学习分布式文件系统奠定了扎实基础。