西安交通大学

操作系统专题实验报告

班级：_____

学号：_____

姓名：_____

年 月 日

目录

1 openEuler 系统环境实验

1.1 实验目的

1.1.1 进程实验

（1）熟悉 Linux 操作系统的基本环境和操作方法，通过运行系统命令查看系统基本信息以了解系统；（2）编写并运行简单的进程调度相关程序，体会进程调度、进程间变量的管理等机制在操作系统实际运行中的作用。

1.1.2 线程实验

探究多线程编程中的线程共享进程信息。在计算机编程中，多线程是一种常见的并发编程方式，允许程序在同一进程内创建多个线程，从而实现并发执行。由于这些线程共享同一进程的资源，包括内存空间和全局变量，因此可能会出现线程共享进程信息的现象。本实验旨在通过创建多个线程并使其共享进程信息，以便深入了解线程共享资源时可能出现的问题。

1.1.3 自旋锁实验

自旋锁作为一种并发控制机制，可以在特定情况下提高多线程程序的性能。本实验旨在通过设计一个多线程的实验环境，以及使用自旋锁来实现线程间的同步，从而实现以下目标：（1）了解自旋锁的基本概念： 通过研究自旋锁的工作原理和特点，深入理解自旋锁相对于其他锁机制的优势和局限性；（2）实验自旋锁的应用： 在一个多线程的实验环境中，设计一个竞争资源的场景，让多个线程同时竞争对该资源的访问；（3）实现自旋锁的同步： 使用自旋锁来保护竞争资源的访问，确保同一时间只有一个线程可以访问该资源，避免数据不一致和竞态条件。

1.2 实验内容

1.2.1 进程实验

（1）熟悉操作命令、编辑、编译、运行程序。完成示例程序的运行验证，多运行几次程序观察结果；去除 wait 后再观察结果并进行理论分析。（2）扩展图 1-1 的程序：a） 添加一个全局变量并在父进程和子进程中对这个变量做不同操作，输出操作结果并解释；b） 在 return 前增加对全局变量的操作并输出结果，观察并解释；c） 修改程序体会在子进程中调用 system 函数和在子进程中调用 exec 族函数；

1.2.2 线程实验

（1）在进程中给一变量赋初值并成功创建两个线程；（2）在两个线程中分别对此变量循环五千次以上做不同的操作（自行设计）并输出结果；（3）多运行几遍程序观察运行结果，如果发现每次运行结果不同，请解释原因并修改程序解决，考虑如何控制互斥和同步；（4）将任务一中第一个实验调用 system 函数和调用 exec 族函数改成在线程中实现，观察运行结果输出进程 PID 与线程 TID 进行比较并说明原因。

1.2.3 自旋锁实验

（1）在进程中给一变量赋初值并成功创建两个线程；（2）在两个线程中分别对此变量循环五千次以上做不同的操作（自行设计）并输出结果；（3）使用自旋锁实现互斥和同步；

1.3 实验思想

1.3.1 进程实验

通过运行示例程序并在此基础上进行实验，验证父子进程之间的关系，学习与进程的创建有关的系统调用功能和作用，结合理论知识解释程序运行的结果，加深对操作系统进程管理的理解。

1.3.2 线程实验

查阅相关资料，通过系统调用的程序接口创建多个线程，并在两个线程中对变量进行不同的操作，观察程序多次运行的结果，用所学理论知识进行解释，加深对线程之间的关系、线程和进程之间的关系与区别的理解。同时对进程（线程）运行的时间片轮转有了初步认识。

1.3.3 自旋锁实验

在线程实验的任务基础上，参考代码框架进行修改，利用自旋锁实现线程之间的同步和互斥，观察程序运行的结果有何不同，验证线程并发执行的过程，学习控制线程并发时的同步和互斥。

1.4 实验步骤

1.4.1 进程实验

本实验通过在程序中输出父、子进程的 pid，分析父子进程 pid 之间的关系，进一步加入 wait()函数分析其作用。

步骤一：编写并多次运行示例中代码。

步骤二：删去代码中的 wait()函数并多次运行程序，分析运行结果。

步骤三：修改代码，增加一个全局变量并在父子进程中对其进行不同的操作（自行设计），观察并解释所做操作和输出结果。

步骤四：在步骤三基础上，在 return 前增加对全局变量的操作（自行设计）并输出结果， 观察并解释所做操作和输出结果。

步骤五：修改程序，在子进程中调用 system()与 exec 族函数。编写 system_call.c 文件输出进程号 PID，编译后生成 system_call 可执行文件。在子进程中调用 system_call,观察输出结果并分析总结。

1.4.2 线程实验

步骤一：设计程序，创建两个子线程，两线程分别对同一个共享变量多次操作，观察输出结果。

步骤二：修改程序，定义信号量 signal，使用 PV 操作实现共享变量的访问与互斥。运行程序，观察最终共享变量的值。

步骤三：在第一部分实验了解了 system()与 exec 族函数的基础上，将这两个函数的调用改为在线程中实现，输出进程 PID 和线程的 TID 进行分析。

1.4.3 自旋锁实验

步骤一：根据实验内容要求，编写模拟自旋锁程序代码 spinlock.c

步骤二：补充完成代码后，编译并运行程序，分析运行结果。

1.5 测试数据设计

(1)进程实验：步骤三，父子进程中要对一个全局变量进行不同的操作。这里我们定义了一个全局变量 value，初始化为 0.在父进程和子进程中分别对其进行一次-1 和+1 的操作。步骤四在此基础上，在 return 前将全局变量 value 重新赋值为 408.

(2)线程实验：两线程要对同一共享变量多次操作。这里初始化变量 value=0，并在两个线程中分别进行 50000 次+1 和 50000 次-1 操作。

(3)自旋锁实验：同样是创建两个线程，共享变量 shared_value，初值为 0.两个线程都是对其进行 5000 次+1 操作。

1.6 程序运行初值及运行结果分析

1.6.1 进程实验

程序运行初值见测试数据设计部分。

多次运行示例程序，结果如下

```
[root@kptest01 文档]# ./ProcessExp1
parent:pid=5333
child:pid=0
parent:pid1=5332
child:pid1=5333
[root@kptest01 文档]# ./ProcessExp1
parent:pid=5335
child:pid=0
parent:pid1=5334
child:pid1=5335
[root@kptest01 文档]# ./ProcessExp1
parent:pid=5337
child:pid=0
parent:pid1=5336
child:pid1=5337
[root@kptest01 文档]# ./ProcessExp1
parent:pid=5339
child:pid=0
parent:pid1=5338
child:pid1=5339
[root@kptest01 文档]#
```

在父进程中通过系统调用 fork()创建子进程，返回的 pid 是不同的。如果是父进程，会返回子进程真正的 id；如果是子进程，则会返回 0。而 getpid()在两个进程中都返回它们各自的 id。可以看出父子进程 pid 并不相同，并且父进程中 fork()返回值与子进程中 getpid()返回值一致。

删去父进程中 wait()，再次运行结果如下：

```
[root@kptest01 文档]# ./ProcessExp1_nowait
parent:pid=5362
child:pid=0
parent:pid1=5361
child:pid1=5362
[root@kptest01 文档]# ./ProcessExp1_nowait
parent:pid=5364
child:pid=0
parent:pid1=5363
child:pid1=5364
[root@kptest01 文档]# ./ProcessExp1_nowait
parent:pid=5366
child:pid=0
parent:pid1=5365
child:pid1=5366
[root@kptest01 文档]#
```

结果看似没有什么区别。事实上，父进程在没有 wait()的情况下，可能先于子进程结束，此时孤儿进程会被系统中一个 init 进程接管。

对全局变量 value（初值 0）在父进程和子进程中分别-1 和+1 操作，打印之后的值：

```
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]#
```

父进程中 value 变为-1，子进程为 1。子进程的地址空间是父进程的一个拷贝，数值相同但其实是两个独立的物理空间。因此它们其实是对各自地址空间的 value 值进行了-1/+1 操作。

在程序代码的 return 前将 value 赋值为 408，打印赋值前、赋值后的结果以及地址信息：

```
[root@kptest01 文档]# ./ProcessExp4
parent:value=-1
child:value=1
It's time to return,value=408,address=4325468
It's time to return,value=408,address=4325468
[root@kptest01 文档]# ./ProcessExp4
parent:value=-1
child:value=1
It's time to return,value=408,address=4325468
It's time to return,value=408,address=4325468
```

最后都输出 408.这个操作在父子进程中都被执行。尽管物理地址独立，但是 value 在各自地址空间中的逻辑地址是一样的。

在子进程中系统调用 exec()族函数或 system()函数，改变子进程要执行的语句。这里子进程重新打印出自己的 pid。

```
[root@kptest01 文档]# ./ProcessExp5_exec
child:pid=0
parent:pid=5480
child:pid1=5480
parent:pid1=5479
After exec/system, the pid is 5480
[root@kptest01 文档]# ./ProcessExp5_exec
parent:pid=5482
child:pid=0
parent:pid1=5481
child:pid1=5482
After exec/system, the pid is 5482
[root@kptest01 文档]# ./ProcessExp5_exec
parent:pid=5485
child:pid=0
parent:pid1=5484
child:pid1=5485
After exec/system, the pid is 5485
```

```
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2486
child:pid=0
parent:pid1=2485
child:pid1=2486
After exec/system, the pid is 2487
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2489
child:pid=0
parent:pid1=2488
child:pid1=2489
After exec/system, the pid is 2490
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2492
child:pid=0
parent:pid1=2491
child:pid1=2492
After exec/system, the pid is 2493
```

子进程调用 execlp/system，前者不会改变进程的 pid 和分配的进程空间，只是替换进程空间里的内容。而后者则会完全创建新的 pid 和进程空间。

1.6.2 线程实验

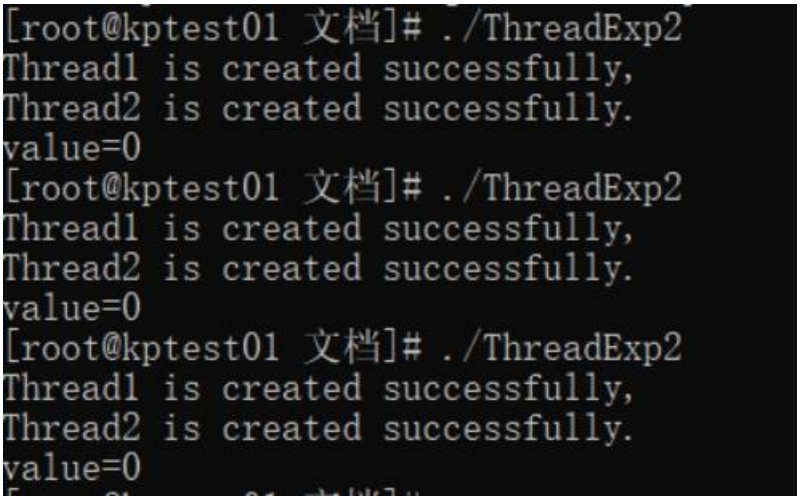
在一个进程中创建两个线程，分别对进程中一个共享变量 value 进行 50000 次+1/-1 操作，多次运行的结果如下

```
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=30090
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=3274
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=2399
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=-2542
```

正常来说我们想得到的结果肯定是 value 值仍然是 0.但是由于线程之间的并发而没有去互斥约束，在它们竞争访问共享变量时，就有可能一个线程进行一次+1 操作后还未写回，另一线程又去取 value 值来做-1 操作。类似这种情况会使得后写回的值覆盖先写回的值，实际上最后的结果就是这样完全随机的，每次都不相同，更不会是预期的值。

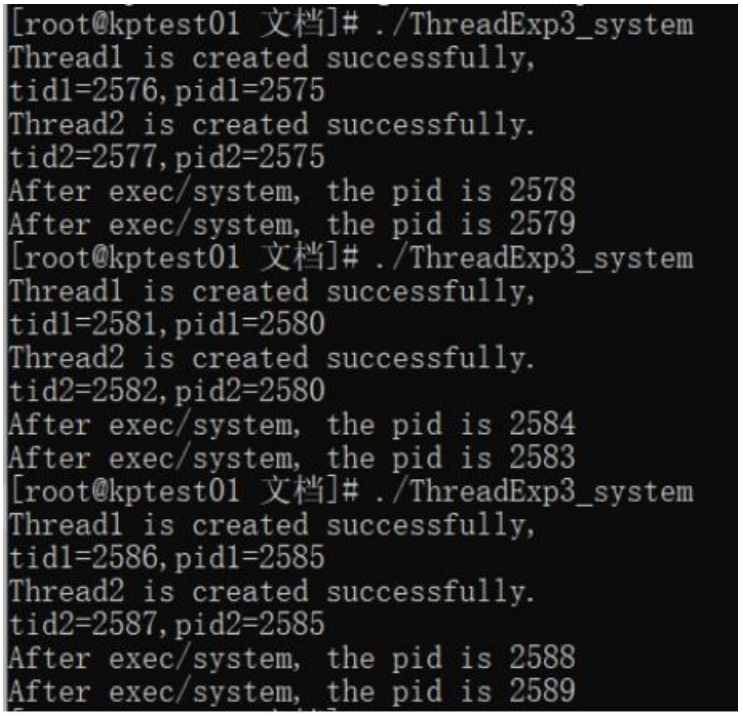
改写程序，定义信号量，使用 PV 操作来实现共享变量的访问和互斥。多次运行程序观察到

最后的结果。



用信号量实现互斥访问，结果稳定输出 0.这是因为互斥访问避免了以上情况。一个线程只有在它完成了一次对共享变量的操作并把它写回内存空间时，该共享变量才能下一次被访问和修改。

在两个线程中打印它们的 pid 和 tid，以及系统调用 system()和 exec()族函数后，结果如下



```
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2553,pid1=2552
Thread2 is created successfully.
tid2=2554,pid2=2552
After exec/system, the pid is 2552
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2562,pid1=2561
Thread2 is created successfully.
tid2=2563,pid2=2561
After exec/system, the pid is 2561
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2565,pid1=2564
Thread2 is created successfully.
tid2=2566,pid2=2564
After exec/system, the pid is 2564
[root@kptest01 文档]#
```

两个线程有同一进程创建，它们没有独立的 pid，共享创建它们的进程的资源。tid 自然是不同的，但是 pid 完全相同。system 调用改变 pid，execlp 不改变。在两个线程中进行这两种调用的结果，和在子进程中调用的结果，对 pid 以及实际上的进程空间的改变是相同的。

1.6.3 自旋锁实验

自旋锁是一种基于忙等待（busy-waiting）的同步机制，用于在线程竞争 共享资源时，不断尝试获取锁，而不是阻塞等待。我们和线程实验中一样在一个进程中创建两个线程，两个线程都是对一个初值为 0 的共享变量进行 5000 次+1 操作，当用自旋锁来实现互斥访问时时，运行结果：

```
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
```

输出结果始终是 10000。

1.7 实验总结

1.7.1 实验中的问题与解决过程

主要是对实验中用到的一些系统调用功能不熟悉，以及进程、线程的创建不熟悉，或是进程、线程创建后的等待、终止不规范解决方法主要是查找相关的资料。见 Readme。

另外，华为云上的 openEuler 环境与本地机略有不同，因此在编译链接产生可执行文件时，用 gcc 命令的参数略有不同。见 Readme。

1.7.2 实验收获

通过实验加深了对进程、线程的相关概念的理解，以及它们之间的关系，知道程序如何创建进程、线程，或是收回进程、线程的资源并终止它们，对进程和线程之间的关系有了很深的印象，对于进程和线程的并发执行有了直观的感受，切实感受到同步和互斥的重要性，并能通过不同的方式对共享变量进行同步和互斥访问。

1.7.3 意见与建议

无。

1.8 附件

1.8.1 附件 1 程序

（1）进程实验，各小步骤只需修改注释掉的行重新生成可执行文件。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>


//int value=0;
int main()
{
    pid_t pid,pid1;
    pid=fork();

    if (pid<0)
    {
        fprintf(stderr,"Fork Failed");
        return 1;
    }
    else if (pid==0)
    {
        pid1=getpid();
        //value++;
        //printf("child:pid=%d\n",pid);
        printf("child:pid1=%d\n",pid1);
        //execlp("/home/hvye/文档/system_call","ls",NULL);
        system("/home/hvye/文档/system_call");
        //printf("child:value=%d\n",value);
```

```
    }
    else
    {
        pid1=getpid();
        //value--;
        //printf("parent:pid=%d\n",pid);
        printf("parent:pid1=%d\n",pid1);
        //printf("parent:value=%d\n",value);
        wait(NULL);
    }
    // value=408;
    // printf("It's time to return,value=%d,address=%d\n",value,&value);
    return 0;
}
```

附 system_call.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
      pid_t new_pid;
      new_pid=getpid();
      printf("After exec/system, the pid is %d\n",new_pid);
      return 0;
}
```

（2）线程实验，各小题也只要修改注释行。

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

int value=0;
sem_t signal;


void* Thread1()
{
    printf("Thread1 is created successfully,\n");
    for (int i=0;i<500000;i++)
    {
        sem_wait(&signal);
        value++;
        sem_post(&signal);

    }
    pthread_t tid1;
    tid1=pthread_self();
    pid_t pid;
    pid=getpid();
```

```
    //printf("tid1=%d,pid1=%d\n",tid1,pid);
    //system("/home/hvye/文档/system_call");
    //execlp("/home/hvye/文档/system_call","ls",NULL);
    pthread_exit(0);
    return NULL;
}

void* Thread2()
{
    printf("Thread2 is created successfully.\n");
    for (int i=0;i<500000;i++)
    {
        sem_wait(&signal);
        value--;
        sem_post(&signal);
    }
     pthread_t tid2;
    tid2=pthread_self();
    pid_t pid;
    pid=getpid();
    //printf("tid2=%d,pid2=%d\n",tid2,pid);
    //system("/home/hvye/文档/system_call");
    //execlp("/home/hvye/文档/system_call","ls",NULL);
    pthread_exit(0);
    return NULL;

}


int main()
{
    pthread_t tid1,tid2;
    sem_init(&signal,0,1);
    pthread_create(&tid1,NULL,Thread1,NULL);
    pthread_create(&tid2,NULL,Thread2,NULL);
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    printf("value=%d\n",value);
    return 0;
}
```

（3）自旋锁实验

```
#include <stdio.h>
#include <pthread.h>

typedef struct
{
      int flag;
}spinlock_t;

void spinlock_init(spinlock_t* lock)
{
      lock->flag=0;
```

```
}

void spinlock_lock(spinlock_t* lock)
{
	while (__sync_lock_test_and_set(&lock->flag,1))
	{

	}//自旋等待
}

void spinlock_unlock(spinlock_t* lock)
{
	__sync_lock_release(&lock->flag);
}

int shared_value=0;

void* thread_function(void* arg)
{
	spinlock_t* lock=(spinlock_t*)arg;
	for (int i=0;i<5000;i++)
	{
		spinlock_lock(lock);
		shared_value++;
		spinlock_unlock(lock);
	}
	return NULL;
}

int main()
{
	pthread_t thread1,thread2;
	spinlock_t lock;
	printf("The initial shared value:%d\n",shared_value);
	spinlock_init(&lock);
	pthread_create(&thread1,NULL,thread_function,(void*)&lock);
	pthread_create(&thread2,NULL,thread_function,(void*)&lock);
	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);
	printf("After thr operation of two threads,the shared
value:%d\n",shared_value);
	return 0;
}
```

1.8.2 附件 2 Readme

system()与 exec 族函数：参考 [system()、exec()、fork()三个与进程有关的函数的比较 - 青儿哥哥 - 博客园]两者都可以在子进程中执行其他的进程，区别是 exec 函数族进行进程体替换，不改变原有的 pid；system 直接创建新的进程，因此会分配新的 pid。两者都会改变进程的数据段和代码段。

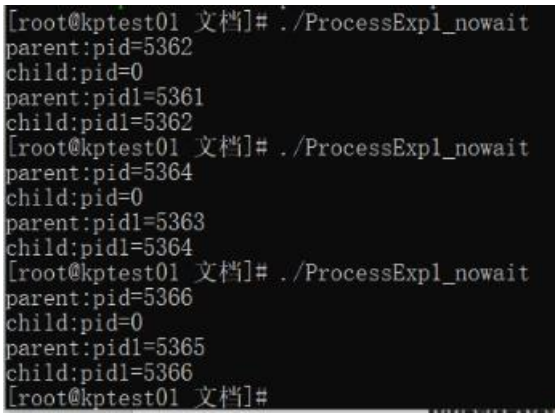
多线程的实现和同步互斥：参考多线程编程（Linux C） - 让我思考一下 - 博客园

两个线程对共享变量多次并发进行操作时，有时多次运行看到终端输相同的结果。这可能是因为 for 循环操作的次数太少，由于时间片很短，没有发生竞争。可以增大操作的次数，结果更明显。

在进程中调用 pthread_create()创建两个线程并发执行后，调试时会发生一种情况，就是有其中一个线程并没有执行。原因是进程在创建完线程之后并没有停下来，因此可能在一个线程还没来得及执行时，进程继续执行完毕并退出，这样进程的全部资源被收回，线程也就消灭了。所以为了确保线程执行完毕，要调用 pthread_join()等待线程执行。

云环境上 Pthread.h 的 reference 报错：参考 Linux 下 undefined reference to ‘pthread_create’ 问题解决-CSDN 博客

1.1 1.2



父子进程中 pid 返回值不同，子进程 pid=0，父进程 pid 为子进程实际的 id。pid1 分别返回父、子进程的实际的 id。因此 parent：pid==child：pid1.

1.3

```
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]# ./ProcessExp3
parent:value=-1
child:value=1
[root@kptest01 文档]#
```

value 初值=0.parent/child 分别--/++，分别输出-1 和 1.父子进程的地址空间是独立的。

1.4

```
[root@kptest01 文档]# ./ProcessExp4
parent:value=-1
child:value=1
It's time to return,value=408,address=4325468
It's time to return,value=408,address=4325468
[root@kptest01 文档]# ./ProcessExp4
parent:value=-1
child:value=1
It's time to return,value=408,address=4325468
It's time to return,value=408,address=4325468
```

return 前都进行赋值 value=408，最后都输出 408.这个操作在父子进程中都被执行。尽管物理地址独立，但是 value 在各自地址空间中的逻辑地址是一样的。

1.5

```
[root@kptest01 文档]# ./ProcessExp5_exec
child:pid=0
parent:pid=5480
child:pid1=5480
parent:pid1=5479
After exec/system, the pid is 5480
[root@kptest01 文档]# ./ProcessExp5_exec
parent:pid=5482
child:pid=0
parent:pid1=5481
child:pid1=5482
After exec/system, the pid is 5482
[root@kptest01 文档]# ./ProcessExp5_exec
parent:pid=5485
child:pid=0
parent:pid1=5484
child:pid1=5485
After exec/system, the pid is 5485
```

```
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2486
child:pid=0
parent:pid1=2485
child:pid1=2486
After exec/system, the pid is 2487
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2489
child:pid=0
parent:pid1=2488
child:pid1=2489
After exec/system, the pid is 2490
[root@kptest01 文档]# ./ProcessExp5_system
parent:pid=2492
child:pid=0
parent:pid1=2491
child:pid1=2492
After exec/system, the pid is 2493
```

子进程调用 execlp/system，前者 pid 不变，后者改变。

2.1

```
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=30090
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=3274
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=2399
[root@kptest01 文档]# ./ThreadExp1
Thread1 is created successfully,
Thread2 is created successfully.
value=-2542
```

两个线程分别对同一全局变量进行多次++/--操作，每次结果不同，这是由于线程之间的并发而没有去互斥约束。

2.2

```
[root@kptest01 文档]# ./ThreadExp2
Thread1 is created successfully,
Thread2 is created successfully.
value=0
[root@kptest01 文档]# ./ThreadExp2
Thread1 is created successfully,
Thread2 is created successfully.
value=0
[root@kptest01 文档]# ./ThreadExp2
Thread1 is created successfully,
Thread2 is created successfully.
value=0
```

用信号量实现互斥访问，结果稳定输出 0.

2.3

```
[root@kptest01 文档]# ./ThreadExp3_system
Thread1 is created successfully,
tid1=2576,pid1=2575
Thread2 is created successfully.
tid2=2577,pid2=2575
After exec/system, the pid is 2578
After exec/system, the pid is 2579
[root@kptest01 文档]# ./ThreadExp3_system
Thread1 is created successfully,
tid1=2581,pid1=2580
Thread2 is created successfully.
tid2=2582,pid2=2580
After exec/system, the pid is 2584
After exec/system, the pid is 2583
[root@kptest01 文档]# ./ThreadExp3_system
Thread1 is created successfully,
tid1=2586,pid1=2585
Thread2 is created successfully.
tid2=2587,pid2=2585
After exec/system, the pid is 2588
After exec/system, the pid is 2589
```

```
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2553,pid1=2552
Thread2 is created successfully.
tid2=2554,pid2=2552
After exec/system, the pid is 2552
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2562,pid1=2561
Thread2 is created successfully.
tid2=2563,pid2=2561
After exec/system, the pid is 2561
[root@kptest01 文档]# ./ThreadExp3_exec
Thread1 is created successfully,
tid1=2565,pid1=2564
Thread2 is created successfully.
tid2=2566,pid2=2564
After exec/system, the pid is 2564
[root@kptest01 文档]#
```

同上，system 调用改变 pid，execlp 不改变。

3.1

```
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
[root@kptest01 文档]# ./spinlock
The initial shared value:0
After thr operation of two threads,the shared value:10000
```

通过自旋锁实现互斥访问，输出结果始终是 10000

2 进程通信与内存管理

2.1 实验目的

2.1.1 软中断通信

编程实现进程的创建和软中断通信，通过观察、分析实验现象，深入理解进程及进程在调度执行和内存空间等方面的特点，掌握在 POSIX 规范中系统调用的功能和使用。

2.1.2 管道通信

编程实现进程的管道通信，通过观察、分析实验现象，深入理解进程管道通信的特点，掌握管道通信的同步和互斥机制。

2.1.3 页面的置换

通过模拟实现页面置换算法（FIFO、LRU），理解请求分页系统中，页面置换的实现思路，理解命中率和缺页率的概念，理解程序的局部性原理，理解虚拟存储的原理。

2.2 实验内容

2.2.1 软中断通信

（1）使用 man 命令查看 fork 、kill 、signal、sleep、exit 系统调用的帮助手册。

（2）根据流程图（如图 2.1 所示）编制实现软中断通信的程序：使用系统调用 fork()创建两个子进程，再用系统调用 signal()让父进程捕捉键盘上发出的中断信号（即 5s 内按下 delete 键或 quit 键），当父进程接收到这两个软中断的某一个后，父进程用系统调用 kill()向两个子进程分别发出整数值为 16 和 17 软中断信号，子进程获得对应软中断信号，然后分别输出下列信息后终止：

Child process 1 is killed by parent !!

Child process 2 is killed by parent !!

父进程调用 wait()函数等待两个子进程终止后，输出以下信息，结束进程执行：

Parent process is killed!!

（3）多次运行所写程序，比较 5s 内按下 Ctrl+\或 Ctrl+Delete 发送中断，或 5s 内不进行任何操作发送中断，分别会出现什么结果？分析原因。

（4）将本实验中通信产生的中断通过 14 号信号值进行闹钟中断，体会不同中断的执行样式，从而对软中断机制有一个更好的理解。

2.2.2 管道通信

（1）学习 man 命令的用法，通过它查看管道创建、同步互斥系统调用的在线帮助，并阅读参考资料。

（2）根据流程图（如图 2.2 所示）和所给管道通信程序，按照注释里的要求把代码补充完整，运行程序，体会互斥锁的作用，比较有锁和无锁程序的运行结果，分析管道通信是如何实现同步与互斥的。

（3）修改上述程序，让其中两个子进程各向管道写入足够多的字符，父进程从管道中读出，分有锁和无锁的情况。运行程序，分别观察有锁和无锁情况下的写入读出情况。

2.2.3 页面的置换

（1）理解页面置换算法 FIFO、LRU 的思想及实现的思路。

（2）定义相应的数据结构，在一个程序中实现上述 2 种算法，运行

时可以选择算法。算法的页面引用序列要至少能够支持随机数自动生成、手动输入两种生成方式；算法要输出页面置换的过程和最终的缺页率。

（3）运行所实现的算法，并通过对比，分析 2 种算法的优劣。

（4）设计测试数据，观察 FIFO 算法的 BLEADY 现象；设计具有局部性特点的测试数据，分别运行实现的 2 种算法，比较缺页率，并进行分析。

2.3 实验思想

2.3.1 软中断通信

通过系统调用函数，用软中断的方式实现通信。在父进程中依次创建两个进程。然后通过产生不同的中断信号值，向子进程通信。可以用系统调用 kill()主动向两个子进程分别发送信号，也可以设置一个等待时间，在等待时间计时结束后通过时钟中断向两个子进程发送相同的闹钟中断信号。子进程一经创建就持续等待监听从父进程传达的中断信号，在接受中断信号后终止进程。父进程在产生中断后等待子进程结束，当两个子进程都终止时，结束父进程。为了观察方便，可以更改用到的中断信号值对应的中断处理函数。

2.3.2 管道通信

所谓“管道”，是指用于连接一个读进程和一个写进程以实现他们之间通信的一个共享文件，又名 pipe 文件。向管道(共享文件)提供输入的发送进程(即写进程)，以字符流形式将大量的数据送入管道；而接受管道输出的接收进程(即读进程)，则从管道中接收(读)数据。由于发送进程和接收进程是利用管道进行通信的，故又称为管道通信。匿名管道只能用于父子进程之间的通信，它的创建使用系统调用 pipe()：

int pipe(int fd[2])

其中的参数 fd 用于描述管道的两端，其中 fd[0]是读端，fd[1]是写端。两个进程分别使用读端和写端，就可以进行通信了。我们在父进程中创建两个子进程，两个子进程分别在管道的写一端写入内容，然后父进程在管道读一端读出内容，观察现象，体会管道通信的思想，并尝试在管道通信中运用互斥和同步的机制。

2.3.3 页面的置换

设计实现数据结构，来模拟虚拟内存的页面置换算法。预先设计好进程的页数和分配内存空间的页数。当进程的页数小于分配的内存页数时，进程运行就可能需要页面置换。主要实现的页面置换算法有：FIFO，即需要页面置换时，最先换出内存中最先装入的那一页：

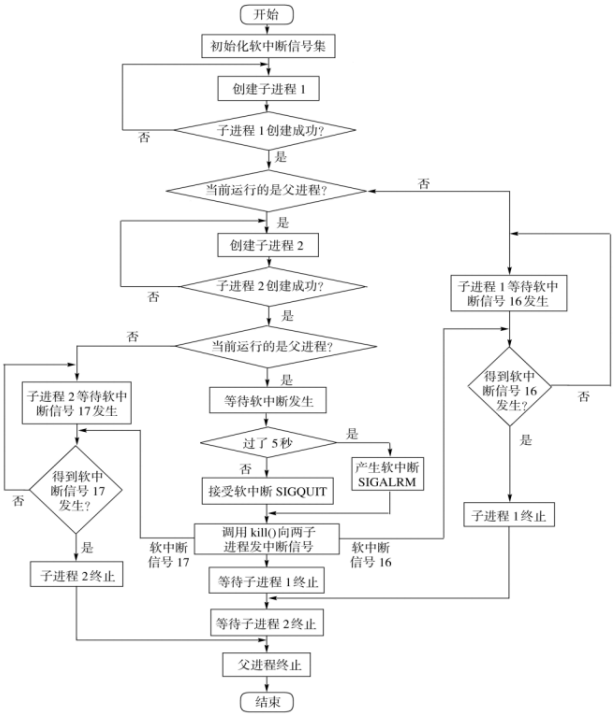
LRU，即需要页面置换时，最先换出此前最近一段时间最少被引用的页。这种算法在进程运行的局部性好时，命中率会更高。

模拟页面置换的过程，并显示每一次置换的过程，最后输出命中率。

2.4 实验步骤

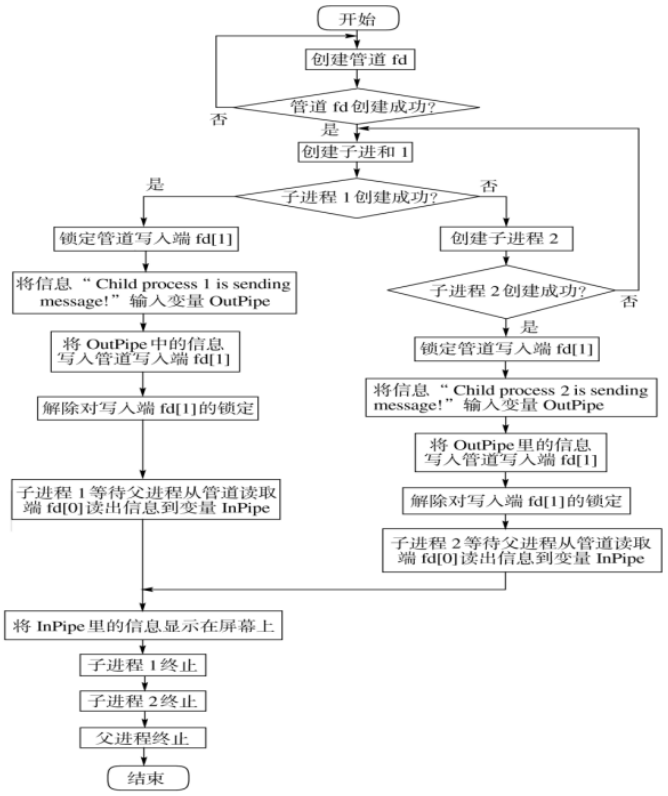
2.4.1 软中断通信

进程软中断通信的实验步骤流程如下：



2.4.2 管道通信

管道通信的实验步骤流程如下



2.4.3 页面的置换

FIFO 算法步骤：

（1）在分配内存页面数（AP）小于进程页面数（PP）时，当然是最先运行的 AP 个页面放入内存；

（2）这时又需要处理新的页面，则将原来放的内存中的 AP 个页中最先进入的调出（FIFO），再将新页面放入；

（3）以后如果再有新页面需要调入，则都按上述规则进行。

LRU 算法步骤：

（1）当内存分配页面数（AP）小于进程页面数（PP）时，把最先执行的 AP 个页面放入内存。

（2）当需调页面进入内存，而当前分配的内存页面全部不空闲时，选择将其中最长时间没有用到的那一页调出，以空出内存来放置新调入的页面（LRU）。

2.5 测试数据设计

主要是对页面置换实验的输入数据设计。我们假设一个程序的页数为 8 而给它分配的内存空间页数只有 3. 由于 LRU 算法在处理具有较好局部性的应用程序时效果会明显更好。据此设计了一个具有较好局部性的进程页调度序列（0-7），依次为：7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1。共 20 次，所谓局部性好，就是一部分页在被访问之后，有很

大的可能会被再次访问，一段时间内会有一部分页会被更频繁地引用。

2.6 程序运行初值及运行结果分析

2.6.1 软中断通信

为便于调试和验证，修改中断处理函数，使得中断发生后打印一个提示信息，提示对应中断信号值对应的中断发生。当父进程 sleep 的 5 秒内在终端 ctrl+c 或者 ctrl+\时，父进程接受到 2 号中断或 3 号中断，通过 kill()系统调用向两个子进程产生 16 号、17 号中断。子进程一经接收中断信号便结束进程。父进程等待子进程均终止后也结束。

在终端手动传递中断信号时，运行结果如下

```
[root@kptest01 exp2]# ./SoftInterruptConnection
^C
2 stop test

17 stop test

16 stop test

Chile process2 is killed by parent.

Child process1 is killed by parent.

Parent process is killed.
```

```
[root@kptest01 exp2]# ./SoftInterruptConnection
^\
3 stop test

17 stop test

Chile process2 is killed by parent.

16 stop test

Child process1 is killed by parent.

Parent process is killed.
```

如果在 sleep 的 5 秒内没有进行任何操作，则产生 14 号时钟中断，之后父进程向子进程发送的中断信号同上。运行结果

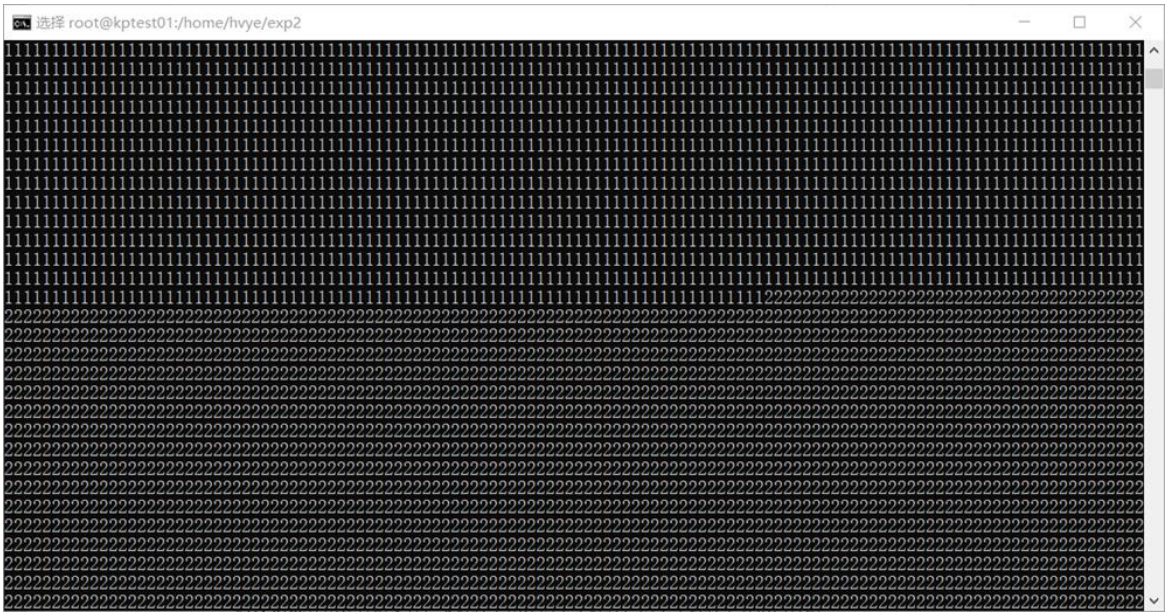
```
[root@kptest01 exp2]# gcc -o SoftInterruptConnection SoftInterruptConnection.c
[root@kptest01 exp2]# ./SoftInterruptConnection

14 stop test

16 stop test

Child process1 is killed by parent.

17 stop test

Chile process2 is killed by parent.

Parent process is killed.
[root@kptest01 exp2]#
```

2.6.2 管道通信

分为有锁和无锁的情况。两个子进程分别重复多次向在管道的读端写入 1 和 2，然后父进程在管道读端读出。当不加锁时，读出的结果如下：

可以看到读出的结果中 1 和 2 无规则交替出现。

当加锁后，两个子进程向管道写入互斥，读出的结果如下



可以看到加锁之后，1 首先被写入管道，写完后 2 才被写入，因此从管道读的一端读出的结果是所有的 1 读完之后再读出所有的 2.

2.6.3 页面置换

页面被引用的序列是 7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1。分配的内存页数为 3，按照 FIFO 算法，这 3 个页帧中装载的进程页号依次为以下过程：

7	7	7	2	2	2	2	4	4	4
	0	0	0	0，hit	3	3	3	2	2
		1	1	1	1	0	0	0	3

0	0	0	0	0	0，hit	0	7	7	7
2	2	2，hit	1	1	1	1，hit	1	0	0
3	3，hit	3	3	2	2	2	2	2	1

程序按此算法进行页面置换的过程：

```
Input 0 for FIFO,1 for LRU
0
Page 7 is loaded at memory block 0.
Page 0 is loaded at memory block 1.
Page 1 is loaded at memory block 2.
Page 2 is swapped in and page 7 is swapped out.
Page 0 is already at memory.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 1 is swapped out.
Page 4 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 4 is swapped out.
Page 3 is already at memory.
Page 2 is already at memory.
Page 1 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 0 is already at memory.
Page 1 is already at memory.
Page 7 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 1 is swapped out.
Page 1 is swapped in and page 2 is swapped out.
Hitted rate:0.250000
```

命中率 25%。

按照 LRU 算法，页面置换过程如下：

7	7	7	2	2	2	2	4	4	4
	0	0	0	0，hit	0	0，hit	0	0	3
		1	1	1	3	3	3	2	2

0	0	0	1	1	1	1，hit	1	1	1，hit
3	3，hit	3	3	3	0	0	0	0，hit	0
2	2	2，hit	2	2，hit	2	2	7	7	7

程序按此算法进行页面置换的过程如下

```
Input 0 for FIFO,1 for LRU
1
Page 7 is loaded at memory block 0.
Page 0 is loaded at memory block 1.
Page 1 is loaded at memory block 2.
Page 2 is swapped in and page 7 is swapped out.
Page 0 is already at memory.
Page 3 is swapped in and page 1 is swapped out.
Page 0 is already at memory.
Page 4 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 4 is swapped out.
Page 3 is already at memory.
Page 2 is already at memory.
Page 1 is swapped in and page 0 is swapped out.
Page 2 is already at memory.
Page 0 is swapped in and page 3 is swapped out.
Page 1 is already at memory.
Page 7 is swapped in and page 2 is swapped out.
Page 0 is already at memory.
Page 1 is already at memory.
Hitted rate:0.400000
```

命中率 40%。

2.7 页面置换算法复杂度分析

在模拟页面置换算法中，用数组的数据结构来模拟分配的页和进程的每个页。设分配的内存页数为 ap，程序的总页数为 pp，数组 page[pp]中记录对应下标的进程页放在内存页的页号，若不在内存中则为-1.pageframe[ap]记录对应下标号的内存页装载的进程页号。例如，page[0]=2 表示进程的第 0 页装载到了内存的第 2 页；pageframe[1]=4 表示内存的第 1 页装载了进程的第 4 页。

对于 FIFO 算法，可以将 pageframe 数组改成一个循环队列。只要用一个整数 head 记录当前内存中最早被装入的那一页帧的页号。根据引用序列检索数组 page[]对应下标的数值，如果不为-1 则命中，否则未命中，替换掉 head 指向的内存页，同时 head 向后移（注意取模）。

对于 LRU 算法，则需要一个等同于分配的页数大小的数组 count[ap]模拟计数器，来记录每一个内存页多久未被引用。判断是否命中的方法同上，不同的是，当命中时，命中的那一个内存页计数清零，其他页计数都+1。当未命中时，如果有空位则不用替换，装入空闲的内存页，内存页计数都+1。如果需要替换，找出当前计数值最大的内存页，它就是最近未使用的页。替换之，计数清零，其他计数+1。

显然 LRU 算法需要额外的计数器数组的存储开销，对应实际情况就是需要额外的硬件开销。在时间上，每一次引用页面，FIFO 算法无论是否命中都只需要 O(1)的时间复杂度。而 LRU 算法无论是否命中，都需要更新、检索计数器数组，因此时间复杂度是 O(ap)。但是，

LRU 在程序局部性较好的情况下命中率更高，可以减少实际系统进行页面置换的许多开销。

2.8 回答问题

2.8.1 软中断通信

（1）猜测的运行结果：在终端操作 ctrl+c 或 ctrl+\产生 SIGINT 或 SIGQUIT 中断，父进程接收后打印提示信息，结束挂起状态向两个子进程发送 16 和 17 号软中断信号。子进程接收到信号打印提示信息后结束，最后父进程结束。如果是没有操作，自动 5 秒后结束挂起状态，产生时钟中断打印提示信息，之后与手动中断相同。不同的中断方式，父进程的打印提示信息不同。

（2）实际结果见运行结果分析部分。接收不同中断，父进程打印的提示信息不同，和预期一致。原因是不同的中断类型对应了不同的中断处理函数。在我们修改的中断处理函数中，打印的信息根据中断类型号是不同的。

（3）改为闹钟中断后，程序在计时器计数完成后结束挂起状态，自动产生一个 14 号中断。不同的是经修改后的中断处理函数打印提示信息，打印的中断信号值不同。

（4）kill()在父进程中调用了两次，分别向两个子进程发送不同的中断信号（16、17），杀死子进程。由于中断信号不同，所以两个子进程在结束前打印的提示信息也不同。

（5）进程在主函数中 return，或调用 exit()都可以主动退出。使用 kill()则是强制性的异常退出。 主动退出更好，若在子进程退出前使用 kill()杀死其父进程，则系统会让 init 进程接管子进程，该子进程就会成为孤儿进程。当使用 kill()杀死子进程使得子进程先于父进程退出时，父进程又没有调用 wait()函数等待子进程结束，子进程处于僵死状态，即僵尸进程。

2.8.2 管道通信

（1）猜测运行结果：不加锁时，读出的 1 和 2 随机无规律出现；如果加锁，那么从管道读出的结果是连续的 1 全部读出后是连续的 2，或者反之。

（2）实际运行结果见运行结果分析部分，和预期相符。不加锁是这种现象的原因是两个子进程竞争管道写的一端，交替向其中写入 1 和 2。加锁之后，当一个进程在写入时，另一个进程就无法访问，直到正在占用的进程写完。

（3）通过 lockf()实现对管道写端的互斥锁，通过父进程等待子进程实现先写后读的同步关系。如果不控制互斥就会出现不加锁时 1 和 2 交替出现的结果，这是由于并发的进程竞争共享资源。如果不控制同步，就有可能父进程在读时子进程没写完甚至管道为空等情况，或是管道已满仍要写入的情况。

2.9 实验总结

2.9.1 实验中的问题与解决过程

主要是对进程之间软中断信号传递的实现，不同中断要使用的系统调用，以及管道等不了解。通过查阅给出的实验指导书以及搜索相关资料学习，以及 man 指令查看帮助解决。

2.9.2 实验收获

通过实践学习了进程之间通信的具体实现方式，概括地说就是消息传递和共享内存，对同步和互斥关系有了更深切的认识，以及学习到不同的实现同步互斥的方式。对两个主要的页面置换算法有了清晰的认识，能描述实现的思想以及替换过程，它们的适用条件及不同的特点、优势和开销代价。

2.9.3 意见与建议

无。

2.10 附件

2.10.1 附件 1 程序

（1）软中断通信，不同的中断只需修改注释行。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <signal.h>
int flag=0;

void inter_handler(int signal)
{
      printf("\n%d stop test\n",signal);
}

void waiting()
{
      signal(2,SIG_IGN);
      signal(3,SIG_IGN);
      pause();
}

int main()
{
      pid_t pid1=-1,pid2=-1;
      while (pid1==-1) pid1=fork();
```

```
	if (pid1>0)
	{
		while (pid2==-1) pid2=fork();
		if (pid2>0)
		{

			signal(14,inter_handler);
			signal(2,inter_handler);
			signal(3,inter_handler);
			alarm(5);
			pause();
			kill(pid2,17);
			kill(pid1,16);
			wait(NULL);
			wait(NULL);
			printf("\nParent process is killed.\n");
		}
		else
		{
			signal(17,inter_handler);
			waiting();
			printf("\nChile process2 is killed by parent.\n");
			return 0;
		}
	}
	else
	{
		signal(16,inter_handler);
		waiting();
		printf("\nChild process1 is killed by parent.\n");
		return 0;
	}
	return 0;
}
```

（2）管道通信

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <stdlib.h>

int pid1,pid2;

int main()
{
	int fd[2];
	char InPipe[30000];
	char* c1="Child process1 is sending message.\n";
	char* c2="Child process2 is senging message.\n";
```

```
	char in1='1';
	char in2='2';
	pipe(fd);
	while ((pid1=fork())==-1) ;
	if (pid1==0)
	{
		lockf(fd[1],1,0);
		//write(fd[1],c1,strlen(c1));
		for (int i=0;i<14000;i++)
		{
			write(fd[1],&in1,1);
		}
		sleep(5);
		lockf(fd[1],0,0);
		exit(0);
	}
	else
	{
		while((pid2=fork())==-1) ;
		if (pid2==0)
		{
			lockf(fd[1],1,0);
			//write(fd[1],c2,strlen(c2)+1);
			for (int i=0;i<14000;i++)
			{
				write(fd[1],&in2,1);
			}
			sleep(5);
			lockf(fd[1],0,0);
			exit(0);
		}
		else
		{
			wait(0);
			wait(0);
			//read(fd[0],InPipe,strlen(c1)+strlen(c2)+1);
			read(fd[0],InPipe,28000);
			InPipe[28001]='\0';
			printf("%s\n",InPipe);
		}


	}
	return 0;
}
```

（3）页面的置换

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
#define total 20 //
#define ap 3 //
#define pp 8 //

// typedef struct qNode
// {
//        int pagenum;
//        qNode* next;
// }qNode;

// qNode createNode()
// {
//        qNode q;
//        q.pagenum = -1;
//        q.next = NULL;
//        return q;
// }

int FIFO_swap(int page[], int pageframe[], int sequence[])
{
   int miss = 0;
   //qNode begin = createNode();
   int head = 0;
   for (int i = 0; i < total; i++)
   {
          if (page[sequence[i]] == -1)
          {
                 miss = miss++;
                 for (int j = 0; j < ap; j++)
                 {
                        if (pageframe[j] == -1)
                        {
                               pageframe[j] = sequence[i];
                               page[sequence[i]] = j;
                               printf("Page %d is loaded at memory block %d.\n",
sequence[i], j);
                               break;
                        }
                        else
                        {
                               if (j == ap - 1)
                               {
                                      printf("Page %d is swapped in and page %d
is swapped out.\n", sequence[i], pageframe[head]);
                                      page[sequence[i]] = head;
                                      page[pageframe[head]] = -1;
                                      pageframe[head] = sequence[i];
                                      head++;
                                      int mod = (int)(head / ap);
                                      head = head - ap * mod;
                               }
```

```
                        }
                    }
                }
                else
                {
                    printf("Page %d is already at memory.\n", sequence[i]);
                }
            }
            return miss;
        }

        int LRU_swap(int page[], int pageframe[], int sequence[])
        {
            int miss = 0;
            int count[ap] = { 0 };
            for (int i = 0; i < total; i++)
            {
                if (page[sequence[i]] == -1)
                {
                    miss++;
                    for (int j = 0; j < ap; j++)
                    {
                        if (pageframe[j] == -1)
                        {
                            pageframe[j] = sequence[i];
                            page[sequence[i]] = j;
                            printf("Page %d is loaded at memory block %d.\n",
sequence[i], j);
                            for (int k = 0; k < ap; k++)
                            {
                                if (pageframe[k] > -1)
                                {
                                    count[k]++;
                                }
                            }
                            count[page[sequence[i]]] = 0;
                            break;
                        }
                        else
                        {
                            if (j == ap - 1)
                            {
                                int idle_time = 0;
                                int least_recent = 0;
                                for (int k = 0; k < ap; k++)
                                {
                                    if (count[k] > idle_time)
                                    {
                                        idle_time = count[k];
                                        least_recent = k;
                                    }
                                }
```

```
                                        printf("Page %d is swapped in and page %d
is swapped out.\n", sequence[i], pageframe[least_recent]);
                                        page[sequence[i]] = least_recent;
                                        page[pageframe[least_recent]] = -1;
                                        pageframe[least_recent] = sequence[i];
                                        for (int k = 0; k < ap; k++)
                                        {
                                               count[k]++;
                                        }
                                        count[least_recent] = 0;
                                 }
                          }
                   }
            }
            else
            {
                   printf("Page %d is already at memory.\n", sequence[i]);
                   for (int j = 0; j < ap; j++)
                   {
                          if (pageframe[j] > -1)
                          {
                                 count[j]++;
                          }
                   }
                   count[page[sequence[i]]] = 0;
            }
      }
      return miss;
   }

   int main()
   {
      int page[pp];
      for (int i = 0; i < pp; i++)
      {
            page[i] = -1;
      }
      int pageframe[ap];
      for (int i = 0; i < ap; i++)
      {
            pageframe[i] = -1;
      }
      int sequence[total];
      printf("Input 0 for random array,1 for yourself.\n");
      int sel1;
      int sel2;
      scanf("%d", &sel1);
      if (sel1 == 1)
      {
            printf("Input 20 page references,from 0 - 7\n");
            for (int i = 0; i < total; i++)
            {
```

```
                scanf("%d", &sequence[i]);
            }
        }
        else if (sel1 == 0)
        {
            srand((unsigned)time(NULL));
            for (int i = 0; i < total; i++)
            {
                sequence[i] = rand() % pp;
            }
        }
        printf("Input 0 for FIFO,1 for LRU\n");
        scanf("%d", &sel2);
        int miss;
        if (sel2 == 0)
        {
            miss = FIFO_swap(page, pageframe, sequence);
        }
        else if (sel2 == 1)
        {
            miss = LRU_swap(page, pageframe, sequence);
        }
        float tmp = 1;
        float hit =tmp-(float)(miss) / (float)(total);
        printf("Hitted rate:%f\n", hit);
        return 0;
    }
```

2.10.2 附件 2 Readme

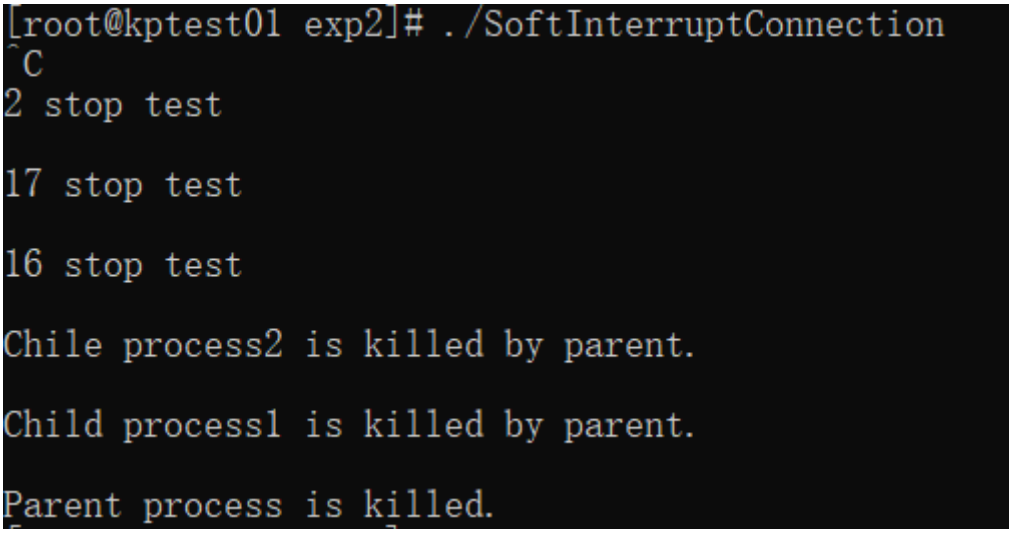
sleep()的系统调用让进程睡眠，但如果睡眠时接收到中断信号会立即结束睡眠，返回 0。为实现计时自动产生中断信号，可以调用 alarm()，计时结束后产生 14 号中断。

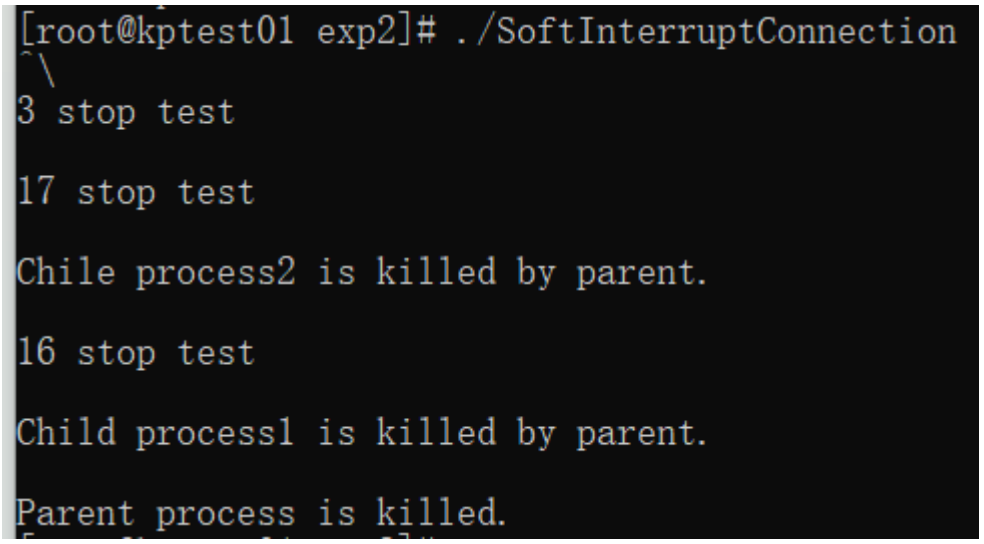
利用信号软中断实现进程通信

主进程中创建两个子进程并计时，子进程等待父进程发送中断信号，一旦接收子进程便被终止。父进程可以通过手动操作发送中断信号，或者不进行任何操作，时间结束后自动发送时钟中断信号。当不进行操作时：

```
[root@kptest01 exp2]# gcc -o SoftInterruptConnection SoftInterruptConnection.c
[root@kptest01 exp2]# ./SoftInterruptConnection

14 stop test

16 stop test

Child process1 is killed by parent.

17 stop test

Chile process2 is killed by parent.

Parent process is killed.
[root@kptest01 exp2]#
```

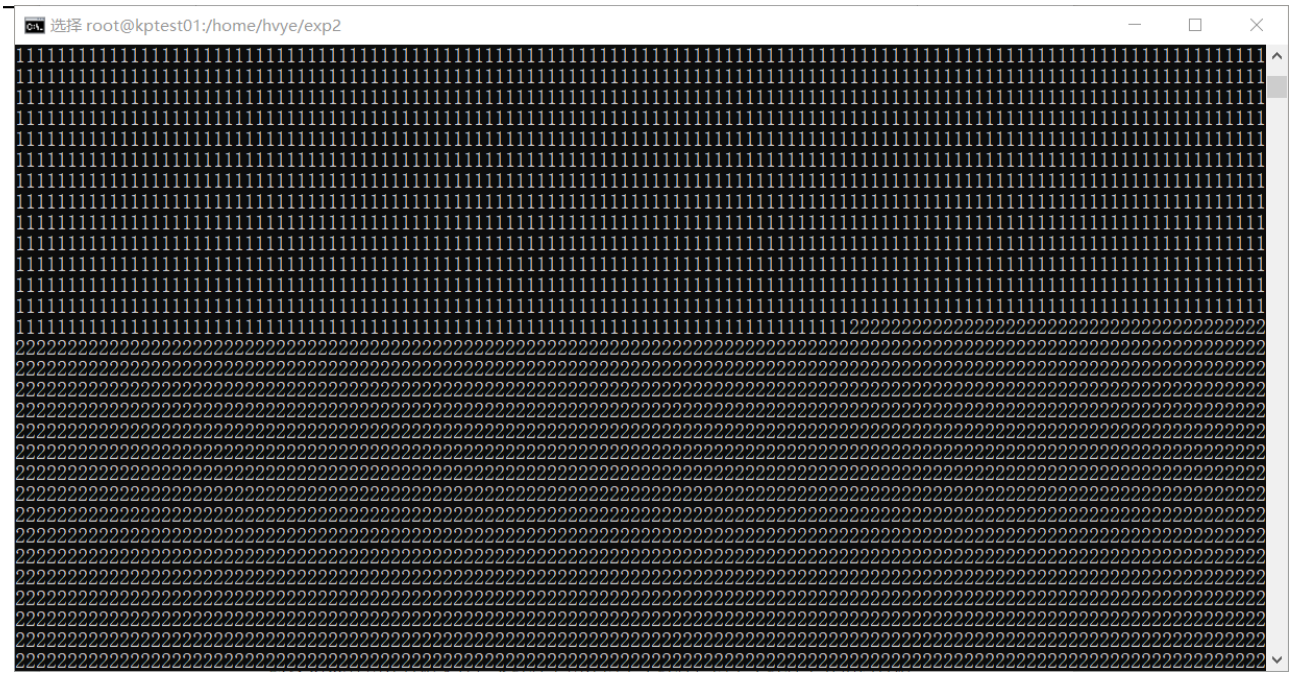
手动操作发送中断信号：





进程之间的管道通信

通过 pipe()创建管道文件，父进程中创建两个子进程，两个子进程中分别向管道写入 1 和 2，写入次数足够多。在有互斥锁时，只有 1 写完才会写 2.



如果没有互斥锁，1 和 2 会随机交替出现。

页面置换算法

FIFO

```
Input 0 for FIFO,1 for LRU
0
Page 7 is loaded at memory block 0.
Page 0 is loaded at memory block 1.
Page 1 is loaded at memory block 2.
Page 2 is swapped in and page 7 is swapped out.
Page 0 is already at memory.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 1 is swapped out.
Page 4 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 4 is swapped out.
Page 3 is already at memory.
Page 2 is already at memory.
Page 1 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 0 is already at memory.
Page 1 is already at memory.
Page 7 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 1 is swapped out.
Page 1 is swapped in and page 2 is swapped out.
Hitted rate:0.250000
```

LRU

```
Input 0 for FIFO,1 for LRU
1
Page 7 is loaded at memory block 0.
Page 0 is loaded at memory block 1.
Page 1 is loaded at memory block 2.
Page 2 is swapped in and page 7 is swapped out.
Page 0 is already at memory.
Page 3 is swapped in and page 1 is swapped out.
Page 0 is already at memory.
Page 4 is swapped in and page 2 is swapped out.
Page 2 is swapped in and page 3 is swapped out.
Page 3 is swapped in and page 0 is swapped out.
Page 0 is swapped in and page 4 is swapped out.
Page 3 is already at memory.
Page 2 is already at memory.
Page 1 is swapped in and page 0 is swapped out.
Page 2 is already at memory.
Page 0 is swapped in and page 3 is swapped out.
Page 1 is already at memory.
Page 7 is swapped in and page 2 is swapped out.
Page 0 is already at memory.
Page 1 is already at memory.
Hitted rate:0.400000
```

3 文件系统

3.1 实验目的

通过一个简单文件系统的设计，理解文件系统的内部结构、基本功能及实现。

3.2 实验内容

（1）分析 EXT2 文件系统的结构；

（2）基于 Ext2 的思想和算法，设计一个类 Ext2 的多级文件系统，实现 Ext2 文件系统的一个功能子集；

（3）用现有操作系统上的文件来代替硬盘进行硬件模拟。

3.3 实验思想

在分析 Linux 的文件系统的基础上，基于 Ext2 的思想和算法，设计一个类 Ext2 的虚拟多级文件系统，实现 Ext2 文件系统的一个功能子集。并且用现有操作系统上的文件来代替硬盘进行硬件模拟。设计文件系统应该考虑的几个层次：①介质的物理结构；②物理操作——设备驱动程序完成；③文件系统的组织结构（逻辑组织结构）；④对组织结构其上的操作；⑤为用户使用文件系统提供的接口。本实验只涉及后三个层次的内容。

3.4 实验步骤

为了进行简单的模拟，基于 Ext2 的思想和算法，设计一个类 Ext2 的文件系统，实现 Ext2 文件系统的一个功能子集。并且用现有操作系统上的文件来代替硬盘进行硬件模拟。

为简单期间，只定义一个组，组描述符只占用一个块，超级块的部分功能也由组描述符的结构体来描述。具体地，主要定义了组描述符、索引节点 inode、以及目录项三种数据结构

```
typedef struct ext2_group_desc
{
	char bg_volumn_name[16];
	int bg_block_bitmap; //块位图的块号
	int bg_inode_bitmap; //索引位图块号
	int bg_inode_table; //索引节点表的起始地址
	int bg_free_blocks_count;
	int bg_free_inodes_count;
	int bg_used_dirs_count;
	char psw[16];
	char bg_pad[24];
}ext2_group_desc;
```

组描述符主要包含以下信息：块位图的块号、索引位图的块号、索引节点表起始地址、空块数、空索引节点数、已使用的目录数，以及用户登录密码。

```
typedef struct ext2_inode
{
	int i_mode; //文件类型和权限
	int i_blocks; //文件数据块个数
	int i_size; //大小
	time_t i_atime;
	time_t i_ctime;
	time_t i_mtime;
```

```
        time_t i_dtime;
        int i_block[8]; //指向数据块的指针
        char i_pad[24];
    }ext2_inode;
```

索引节点主要记录了文件描述信息，包括文件类型、占用的数据块数、文件大小、最后一次访问时间、创建时间、修改时间，以及最重要的文件地址。

```
    typedef struct   ext2_dir_entry
    {
        int inode;
        int rec_len; //目录项长度
        int name_len; //文件名长度
        int file_type;
        char name[EXT2_NAME_LEN];
        char dir_pad;
    }ext2_dir_entry;
```

目录项主要记录文件名，以及文件的索引节点号，它们时用来实现按名查找的关键。其余还有文件类型、文件名长度等。

用一个文件模拟磁盘，一个块大小 512B。第一个块放组描述符，接下来两个块分别为数据块位图和索引节点位图。因此，索引节点和物理块数都是 512*8=4096。由于每个索引结点大小为 64 个字节，最多有 512*8=4096 个索引结点。故，索引结点表的大小为 64*4096=256KB，512 个块。为了和 ext2 保持一致，索引结点从 1 开始计数，0 表示 NULL。数据块则从 0 开始计数。基于以上若干定义，得到模拟文件系统的“硬盘”数据结构

组描述符	数据块位图	索引节点位图	索引节点表	数据块
1 块	1 块	1 块	512 块	4096 块

文件系统将目录体当作一种文件类型进行处理。同时采用混合索引结构，索引节点中 i_block 数组大小为 8，前六个作为直接索引，直接作为一个文件前 6 个物理块的指针。i_block[6] 作为一级索引，指向一个一级索引块，由一个 int 类型 4B 可知，一个一级索引块可以记录 512/4=128 个物理块地址。其余的物理块用作二级索引，即 i_block[7]指向一个物理块，这个物理块记录了 128 个整数，每个整数再指向一个物理块，记录文件块所在物理地址，类似于间接寻址的嵌套。据此，就可以根据文件中一个块的块号，找到所在的物理块号。

文件系统的一个核心功能是实现按名查找文件。根据以上数据结构，按名查找的大致过程是：在当前目录（这也是一个文件，前面已经说明）依次查找每一个目录项的结构体，匹配目录项名，当匹配成功时就能找到这个文件，根据目录项结构体中的索引节点号，找到索引节点在索引节点块中的位置，再根据索引节点结构体中的块指针数组 i_block[8]，找到这个文件的每一个块。文件的块个数和大小也在索引节点中有记录。

用文件模拟磁盘进行操作，也就是不断根据结构体所记录的数据，按照上述过程进行计算得到对应的物理地址，移动文件指针。

根据以上按名查找的思路，主要实现的函数功能如下：

format()：初始化文件系统，主要是初始化组描述符（第一个块）以及根目录（数据块的第

一个块，同时也要分配索引节点，所以数据块位图和索引节点位图的第一位初始化为 1）。除了这些有初始值之外，模拟磁盘的其他内容都填 0。

login()：用户登录，获取终端输入的字符，与组描述符中 password 对比即可。

password()：修改登录密码，登录成功后，从终端获取用户输入的字符，然后更改组描述符结构体中记录的密码。

cd()：进入一个目录。获得在终端的输入，在当前目录按名检索，查找到同名的目录后，就根据上述按名查找过程找到该目录所在的物理地址，然后当前的移动文件指针。

ls()：列表当前目录。按上述按名查找方式找到当前目录所在的物理地址，根据目录大小和一个目录项的大小可以得出目录项的个数，然后依次移动文件指针遍历每一个目录项结构体并打印输出。

read()：向文件写入。按上述按名查找方式找到同名文件所在的物理地址，移动文件指针，然后用文件写入的方式写内容。

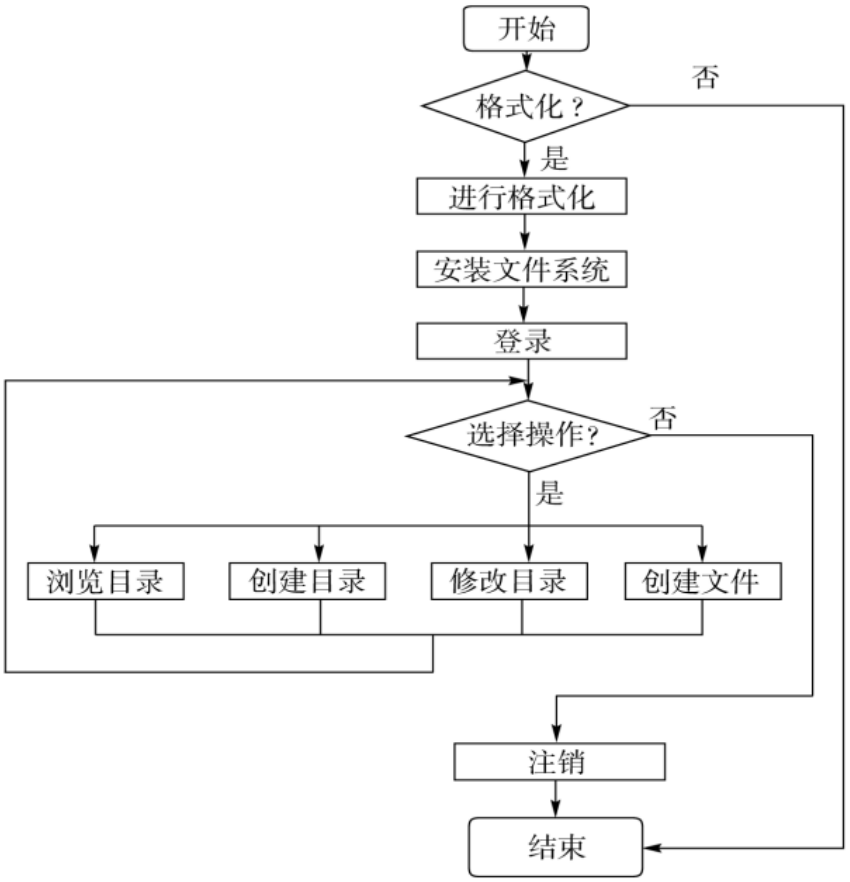
write()：读取文件。类似写入，只不过用读文件的方式读取内容。要进行块扩展时，要更新对应索引节点信息以及找到空闲块，更新块位图，

create()：创建一个文件。在索引节点位图和块位图中找到空闲的索引节点和空闲块、索引节点号和起始块号并更新两个位图。然后初始化索引节点，以及在当前目录下添加一个同名目录项，同时更改当前目录的有关信息描述，如当前目录索引节点中记录的文件大小、文件类型等。

delete()：删除一个文件。如果是普通文件，按名查找找到它的索引节点，由索引节点的 i_block[]知道它的所有物理块号。只要更改两个位图，即可表明对应索引节点或者块为空。然后删去当前目录下的这一目录项，同时要更新当前目录的描述信息。如果要删除的是空目录，则与删除文件类似。如果目录非空，就需要再遍历依次删去这个目录里的所有内容，即递归地删除。这里的空目录并非完全空，因为每一个目录在创建时会初始化两个目录项，分别是当前目录和上一级目录，这是文件系统常用的快速回到上一级操作。

3.5 程序运行初值及运行结果分析

按照以下流程对文件系统进行简单的测试：



测试结果：

```
[root@kptest01 exp3]# ./EXT2_like_FileSystem
Welcome!
File system:Not Found.Create one?
[Y/N]Y
Input password(init:123):123

/=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .     Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024
Directory ..    Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024

/=>#create d folder1
folder1 created.

/=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .     Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024
Directory ..    Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024
Directory folder1       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024        Wed Dec  4 17:06:48 2024

/=>#
```

```
/=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .     Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024
Directory ..    Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024
Directory folder1       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024        Wed Dec  4 17:06:48 2024

/=>#cd folder1

folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .     Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024        Wed Dec  4 17:06:48 2024
Directory ..    Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024        Wed Dec  4 17:06:24 2024

folder1=>#create f file1
file1 created.

folder1=>#create f file2
file2 created.

folder1=>#write file1

Hello world!

folder1=>#
```

```
folder1=>#write file1

Hello world!

folder1=>#write file2

Good evening!


folder1=>#read file1

Hello world!

folder1=>#read file2

Good evening!


folder1=>#delete f file1
file1 deleted.

folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .      Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024
Directory ..     Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024
File file2       Wed Dec  4 17:07:47 2024       Wed Dec  4 17:08:55 2024       Wed Dec  4 17:08:47 2024

folder1=>#
```

3.6 实验总结

3.6.1 实验中的问题与解决过程

遇到的问题：

1. 类 ext2 文件系统的基本功能对应的命令的作用各不相同实现的细节较为繁琐

2. 用文件模拟磁盘进行操作时，是一个二进制文件，像磁盘一样对程序员不可见，因此当输出结果不对时，不方便通过打印的方式定位错误的原因

3. 各模块之间调用关系嵌套，给调试带来困难

尝试解决的对策：

1. 用模块化设计的方式，逐个实现要完成的功能，经过调试确保无误。例如，先实现简单的 format(),login(),logout()，然后根据系统测试的流程，依此实现 ls，cd，create，write，read，delete 等功能。

2. 为处理多个方法调用嵌套为 debug 带来的阻碍，可以将不同的功能先在不同的源代码中实现，再集成

3. 分别调试不同功能的时候，就可以打断点调试查错

4. 仔细思考磁盘逻辑分区的方法和细节，做好笔记。模拟磁盘不同分区的读写通过文件指针的移动来实现，数学计算比较复杂，尤其是涉及混合索引的部分要仔细检查。

3.6.2 实验收获

对文件系统的结构和功能有了非常明确的认识和巩固，理解文件系统对物理磁盘进行管理的思想，极大地锻炼了实践能力，积累了对大量模块进行集成的经验和对大规模程序进行调和和设计的思路。意见与建议

无。

3.7 附件

3.7.1 附件 1 程序

```
#include <stdio.h>
#include "string.h"
#include "stdlib.h"
#include "time.h"
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>
#define blocks 4611 //总块数
#define blocksiz 512 //块大小
#define inodesiz 64 //索引节点长度
#define data_begin_block 515 //数据开始块
#define dirsiz 32 //目录长度
#define EXT2_NAME_LEN 15 //文件名长度
#define PATH "vdisk"  //文件系统

typedef struct ext2_group_desc
{
    char bg_volumn_name[16];
    int bg_block_bitmap; //块位图的块号
    int bg_inode_bitmap; //索引位图块号
    int bg_inode_table; //索引节点表的起始地址
    int bg_free_blocks_count;
    int bg_free_inodes_count;
    int bg_used_dirs_count;
    char psw[16];
    char bg_pad[24];
}ext2_group_desc;

typedef struct ext2_inode
{
    int i_mode; //文件类型和权限
    int i_blocks; //文件数据块个数
    int i_size; //大小
    time_t i_atime;
    time_t i_ctime;
    time_t i_mtime;
    time_t i_dtime;
    int i_block[8]; //指向数据块的指针
    char i_pad[24];
}ext2_inode;

typedef struct  ext2_dir_entry
{
    int inode;
    int rec_len; //目录项长度
```

```
    int name_len; //文件名长度
    int file_type;
    char name[EXT2_NAME_LEN];
    char dir_pad;
}ext2_dir_entry;

ext2_group_desc group_desc;
ext2_inode inode;
ext2_dir_entry dir;
FILE* f;
unsigned int last_alloc_inode=0;
unsigned int last_alloc_block=0;

int getch() //读取键盘输入字符，不回显
{
    int ch;
    struct termios oldt,newt;
    tcgetattr(STDIN_FILENO,&oldt);
    newt=oldt;//保存终端设置
    newt.c_lflag&=~(ECHO|ICANON);//更改终端设置，俄读输入不回显
    tcsetattr(STDIN_FILENO,TCSANOW,&newt);
    ch=getchar();
    tcsetattr(STDIN_FILENO,TCSANOW,&oldt);//恢复终端设置
    return ch;
}

//在当前目录建立一个 vdisk 文件，模拟磁盘
int format()
{
    FILE* fp=NULL;
    int i;
    unsigned int zero[blocksiz/4];
    time_t now;
    time(&now); //计算当前日历时间，并把它编码成 time_t 格式。
    //初始化整个磁盘里所有块，全 0
    while (fp==NULL)
    {
        fp=fopen(PATH,"w+");
    }
    for (i=0;i<blocksiz/4;i++)
    {
        zero[i]=0;
    }
    for (i=0;i<blocks;i++)
    {
        fseek(fp,i*blocksiz,SEEK_SET);
        fwrite(&zero,blocksiz,1,fp);
    }
    //初始化组描述符
    strcpy(group_desc.bg_volumn_name,"user");
    group_desc.bg_block_bitmap=1;
    group_desc.bg_inode_bitmap=2;
```

```
    group_desc.bg_inode_table=3;
    group_desc.bg_free_blocks_count=4095;
    group_desc.bg_free_inodes_count=4095;
    group_desc.bg_used_dirs_count=1;
    strcpy(group_desc.psw,"123");
    fseek(fp,0,SEEK_SET);
    fwrite(&group_desc,sizeof(ext2_group_desc),1,fp);
    //初始化索引节点和块的不位图
    zero[0]=0x80000000;
    fseek(fp,1*blocksiz,SEEK_SET);
    fwrite(&zero,blocksiz,1,fp);
    fseek(fp,2*blocksiz,SEEK_SET);
    fwrite(&zero,blocksiz,1,fp);
    //初始化第一个索引节点
    inode.i_mode=2;
    inode.i_blocks=1;
    inode.i_size=64;
    inode.i_ctime=now;
    inode.i_atime=now;
    inode.i_mtime=now;
    inode.i_dtime=0;
    fseek(fp,3*blocksiz,SEEK_SET);
    fwrite(&inode,sizeof(ext2_inode),1,fp);
    //第一个数据块写当前目录
    dir.inode=0;
    dir.rec_len=32;
    dir.name_len=1;
    dir.file_type=2;
    strcpy(dir.name,".");
    fseek(fp,blocksiz*data_begin_block,SEEK_SET);
    fwrite(&dir,sizeof(ext2_dir_entry),1,fp);
    //当前目录块之后 是上一级目录
    dir.inode=0;
    dir.rec_len=32;
    dir.name_len=2;
    dir.file_type=2;
    strcpy(dir.name,"..");
    fseek(fp,blocksiz*data_begin_block+dirsiz,SEEK_SET);
    fwrite(&dir,sizeof(ext2_dir_entry),1,fp);

    fclose(fp);
    return 0;
}

//将一个目录相对数据块的起始址（当前目录）转换为绝对地址
int dir_entry_position(int dir_entry_begin,int i_block[8])
{
    int dir_blocks=dir_entry_begin/512;
    int block_offset=dir_entry_begin%512;
    int a;
    FILE* fp=NULL;
    if (dir_blocks<=6)
```

```
    {
        return data_begin_block*blocksiz+i_block[dir_blocks]*blocksiz+block_offset;
    }
    else
    {
        while(fp==NULL)
        {
            fp=fopen(PATH,"r+");
        }
        dir_blocks-=6;
        if (dir_blocks<=128)
        {
            int a;

fseek(fp,data_begin_block*blocksiz+i_block[6]*blocksiz+dir_blocks*4,SEEK_SET);
            fread(&a,sizeof(int),1,fp);
            return data_begin_block*blocksiz+a*blocksiz+block_offset;
        }
        else
        {
            dir_blocks-=128;

fseek(fp,data_begin_block*blocksiz+i_block[7]*blocksiz+dir_blocks/128*4,SEEK_SET);
            fread(&a,sizeof(int),1,fp);

fseek(fp,data_begin_block*blocksiz+a*blocksiz+dir_blocks%128*4,SEEK_SET);
            fread(&a,sizeof(int),1,fp);
            return data_begin_block*blocksiz+a*blocksiz+block_offset;
        }
    }
    fclose(fp);
}
//在当前目录打开一个目录项，current 将成为该目录的 inode
int Open(ext2_inode* current,char* name)
{
    FILE* fp=NULL;
    int i;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
    for (i=0;i<(current->i_size/32);i++)
    {
        fseek(fp,dir_entry_position(i*32,current->i_block),SEEK_SET);
        fread(&dir,sizeof(ext2_dir_entry),1,fp);
        if (!strcmp(dir.name,name))
        {
            if (dir.file_type==2)
            {
                //如果要打开的三名是目录体，将 current 指向其 inode
                fseek(fp,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
                fread(current,sizeof(ext2_inode),1,fp);
```

```
                fclose(fp);
                return 0;
            }
        }
    }
    fclose(fp);
    return 1;
}

int Close(ext2_inode* current)
{
    time_t now;
    ext2_dir_entry bentry;
    FILE* fout;
    fout=fopen(PATH,"r+");
    time(&now);
    current->i_atime=now;
    fseek(fout,(data_begin_block+current->i_block[0]*blocksiz),SEEK_SET);
    fread(&bentry,sizeof(ext2_dir_entry),1,fout);
    fseek(fout,3*blocksiz+(bentry.inode)*sizeof(ext2_inode),SEEK_SET);
    fwrite(current,sizeof(ext2_inode),1,fout);
    fclose(fout);
    return Open(current,"..");
}

int Read(ext2_inode* current,char* name)
{
    FILE* fp=NULL;
    int i;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
    for (i=0;i<(current->i_size/32);i++)
    {
        fseek(fp,dir_entry_position(i*32,current->i_block),SEEK_SET);
        fread(&dir,sizeof(ext2_dir_entry),1,fp);
        if (!strcmp(dir.name,name))
        {
            if (dir.file_type==1)
            {
                time_t now;
                ext2_inode node;
                char content_char;
                fseek(fp,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
                fread(&node,sizeof(ext2_inode),1,fp);
                i=0;
                for (i=0;i<node.i_size;i++)
                {
                    fseek(fp,dir_entry_position(i,node.i_block),SEEK_SET);
                    fread(&content_char,sizeof(char),1,fp);
                    if (content_char==0xD) printf("\n");
```

```
                    else printf("%c",content_char);
                }
                printf("\n");
                time(&now);
                node.i_atime=now;
                fseek(fp,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
                fwrite(&node,sizeof(ext2_inode),1,fp);
                fclose(fp);
                return 0;
            }
        }
    }
    fclose(fp);
    return 1;
}
//查找空 inode，返回在 bitmap 中的位置
int FindInode()
{
    FILE* fp=NULL;
    unsigned int zero[blocksiz/4];
    int i;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
    fseek(fp,2*blocksiz,SEEK_SET);
    fread(zero,blocksiz,1,fp);
    for (i=last_alloc_inode;i<(last_alloc_inode+blocksiz/4);i++)
    {
        if (zero[i%(blocksiz/4)]!=0xffffffff)
        {
            unsigned int j=0x80000000,k=zero[i%(blocksiz/4)],l=i;
            for (i=0;i<32;i++)
            {
                if (!(k&j))
                {
                    zero[l%(blocksiz/4)]=zero[l%(blocksiz/4)]|j;
                    group_desc.bg_free_inodes_count--=1;
                    fseek(fp,0,0);
                    fwrite(&group_desc,sizeof(ext2_group_desc),1,fp);
                    fseek(fp,2*blocksiz,SEEK_SET);
                    fwrite(zero,blocksiz,1,fp);
                    last_alloc_inode=l%(blocksiz/4);
                    fclose(fp);
                    return l%(blocksiz/4)*32+i;
                }
                else
                {
                    j=j/2;
                }
            }
        }
```

```
    }
    fclose(fp);
    return -1;
}
//查找空 block，返回在 bitmap 中的位置
int FindBlock()
{
    FILE* fp=NULL;
    unsigned int zero[blocksiz/4];
    int i;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
    fseek(fp,1*blocksiz,SEEK_SET);
    fread(zero,blocksiz,1,fp);
    for (i=last_alloc_block;i<(last_alloc_block+blocksiz/4);i++)
    {
        if (zero[i%(blocksiz/4)]!=0xffffffff)
        {
            unsigned int j=0x80000000,k=zero[i%(blocksiz/4)],l=i;
            for (i=0;i<32;i++)
            {
                if (!(k&j))
                {
                    zero[l%(blocksiz/4)]=zero[l%(blocksiz/4)]|j;
                    group_desc.bg_free_blocks_count-=1;
                    fseek(fp,0,0);
                    fwrite(&group_desc,sizeof(ext2_group_desc),1,fp);
                    fseek(fp,1*blocksiz,SEEK_SET);
                    fwrite(zero,blocksiz,1,fp);
                    last_alloc_block=l%(blocksiz/4);
                    fclose(fp);
                    return l%(blocksiz/4)*32+i;
                }
                else
                {
                    j=j/2;
                }
            }
        }
    }
    fclose(fp);
    return -1;
}

void DelInode(int len)
{
    unsigned int zero[blocksiz/4],i;
    int j;
    f=fopen(PATH,"r+");
    fseek(f,2*blocksiz,SEEK_SET);
```

```
    fread(zero,blocksiz,1,f);
    i=0x80000000;
    for (j=0;j<len%32;j++)
    {
        i=i/2;
    }
    zero[len/32]=zero[len/32]^i;
    fseek(f,2*blocksiz,SEEK_SET);
    fwrite(zero,blocksiz,1,f);
    fclose(f);
}

void DelBlock(int len)
{
    unsigned int zero[blocksiz/4],i;
    int j;
    f=fopen(PATH,"r+");
    fseek(f,1*blocksiz,SEEK_SET);
    fread(zero,blocksiz,1,f);
    i=0x80000000;
    for (j=0;j<len%32;j++)
    {
        i=i/2;
    }
    zero[len/32]=zero[len/32]^i;
    fseek(f,1*blocksiz,SEEK_SET);
    fwrite(zero,blocksiz,1,f);
    fclose(f);
}

void add_block(ext2_inode* current,int i,int j)
{
    FILE* fp=NULL;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
        if (i<6)
        {
            current->i_block[i]=j;
        }
        else
        {
            i-=6;
            if (i==0)
            {
                current->i_block[6]==FindBlock();
                fseek(fp,data_begin_block*blocksiz+current-
>i_block[6]*blocksiz,SEEK_SET);
                fwrite(&j,sizeof(int),1,fp);
            }
            else if (i<128)
```

```
            {
                fseek(fp,data_begin_block*blocksiz+current-
>i_block[6]*blocksiz+i*4,SEEK_SET);
                fwrite(&j,sizeof(int),1,fp);
            }
            else
            {
                i-=128;
                if (i==0)
                {
                    current->i_block[7]=FindBlock();
                    fseek(fp,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz,SEEK_SET);
                    i=FindBlock();
                    fwrite(&i,sizeof(int),1,fp);
                    fseek(fp,data_begin_block*blocksiz+i*blocksiz,SEEK_SET);
                    fwrite(&j,sizeof(int),1,fp);
                }
                if (i%128==0)
                {
                    fseek(fp,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz+i/128*4,SEEK_SET);
                    i=FindBlock();
                    fwrite(&i,sizeof(int),1,fp);
                    fseek(fp,data_begin_block*blocksiz+i*blocksiz,SEEK_SET);
                    fwrite(&j,sizeof(int),1,fp);
                }
                else
                {
                    fseek(fp,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz+i/128*4,SEEK_SET);
                    fread(&i,sizeof(int),1,fp);
                    fseek(fp,data_begin_block*blocksiz+i*blocksiz+i%128*4,SEEK_SET);
                    fwrite(&j,sizeof(int),1,fp);
                }
            }
        }

}


//存疑
int FindEntry(ext2_inode* current)
{
    FILE* fout=NULL;
    int location;
    int block_location;
    int temp;
    int remain_block;
    location=data_begin_block*blocksiz;
    temp=blocksiz/sizeof(int);
    fout=fopen(PATH,"r+");
```

```
    if (current->i_size%blocksiz==0)
    {
        add_block(current,current->i_blocks,FindBlock());
        current->i_blocks++;
    }
    if (current->i_blocks<6)
    {
        location+=current->i_block[current->i_blocks-1]*blocksiz;
        location+=current->i_size%blocksiz;
    }
    else if (current->i_blocks<temp+5)
    {
        block_location=current->i_block[6];
        fseek(fout,(data_begin_block+block_location)*blocksiz+(current->i_blocks-
6)*sizeof(int),SEEK_SET);
        fread(&block_location,sizeof(int),1,fout);
        location+=block_location*blocksiz;
        location+=current->i_size%blocksiz;
    }
    else
    {
        block_location=current->i_block[7];
        remain_block=current->i_blocks-6-temp;
        fseek(fout,(data_begin_block+block_location)*blocksiz+(int)((remain_block-
1)/temp+1)*sizeof(int),SEEK_SET);
        fread(&block_location,sizeof(int),1,fout);
        remain_block=remain_block%temp;

fseek(fout,(data_begin_block+block_location)*blocksiz+remain_block*sizeof(int),SEEK
_SET);
        fread(&block_location,sizeof(int),1,fout);
        location+=block_location*blocksiz;
        location+=current->i_size%blocksiz+dirsiz;
    }
    current->i_size+=dirsiz;
    fclose(fout);
    return location;
}

int Create(int type,ext2_inode* current,char* name)
{
    FILE* fout=NULL;
    int i;
    int block_location;
    int node_location;
    int dir_entry_location;

    time_t now;
    ext2_inode ainode;
    ext2_dir_entry aentry,bentry;
    time(&now);
    fout=fopen(PATH,"r+");
```

```
    node_location=FindInode();
    for (i=0;i<current->i_size/dirsiz;i++)
    {
        fseek(fout,dir_entry_position(i*sizeof(ext2_dir_entry),current-
>i_block),SEEK_SET);
        fread(&aentry,sizeof(ext2_dir_entry),1,fout);
        if (aentry.file_type==type&&!strcmp(aentry.name,name)) return 1;
    }
    fseek(fout,(data_begin_block+current->i_block[0])*blocksiz,SEEK_SET);
    fread(&bentry,sizeof(ext2_dir_entry),1,fout);
    if (type==1)
    {
        ainode.i_mode=1;
        ainode.i_blocks=0;
        ainode.i_size=0;
        ainode.i_ctime=now;
        ainode.i_atime=now;
        ainode.i_mtime=now;
        ainode.i_dtime=0;
        for (i=0;i<8;i++) ainode.i_block[i]=0;
        for (i=0;i<24;i++) ainode.i_pad[i]=0;
    }
    else
    {
        ainode.i_mode=2;
        ainode.i_blocks=1;
        ainode.i_size=64;
        ainode.i_ctime=now;
        ainode.i_atime=now;
        ainode.i_mtime=now;
        ainode.i_dtime=0;
        block_location=FindBlock();
        ainode.i_block[0]=block_location;
        for (i=1;i<8;i++) ainode.i_block[i]=0;
        for (i=0;i<24;i++) ainode.i_pad[i]=(char)(0xff);
        aentry.inode=node_location;
        aentry.rec_len=sizeof(ext2_dir_entry);
        aentry.name_len=1;
        aentry.file_type=2;
        strcpy(aentry.name,".");
        aentry.dir_pad=0;
        fseek(fout,(data_begin_block+block_location)*blocksiz,SEEK_SET);
        fwrite(&aentry,sizeof(ext2_dir_entry),1,fout);
        aentry.inode=bentry.inode;
        aentry.rec_len=sizeof(ext2_dir_entry);
        aentry.name_len=2;
        aentry.file_type=2;
        strcpy(aentry.name,"..");
        aentry.dir_pad=0;
        fwrite(&aentry,sizeof(ext2_dir_entry),1,fout);
        aentry.inode=0;
        aentry.rec_len=sizeof(ext2_dir_entry);
```

```
        aentry.name_len=0;
        aentry.file_type=0;
        aentry.name[EXT2_NAME_LEN]=0;
        aentry.dir_pad=0;
        fwrite(&aentry,sizeof(ext2_dir_entry),14,fout);
    }
    fseek(fout,3*blocksiz+(node_location)*sizeof(ext2_inode),SEEK_SET);
    fwrite(&ainode,sizeof(ext2_inode),1,fout);
    aentry.inode=node_location;
    aentry.rec_len=dirsiz;
    aentry.name_len=strlen(name);
    if (type==1) aentry.file_type=1;
    else aentry.file_type=2;
    strcpy(aentry.name,name);
    aentry.dir_pad=0;
    dir_entry_location=FindEntry(current);
    fseek(fout,dir_entry_location,SEEK_SET);
    fwrite(&aentry,sizeof(ext2_dir_entry),1,fout);
    fseek(fout,3*blocksiz+(bentry.inode)*sizeof(ext2_inode),SEEK_SET);
    //printf("When create,size:%d\n",current->i_size);
    fwrite(current,sizeof(ext2_inode),1,fout);
    fclose(fout);
    return 0;
}

int Write(ext2_inode* current,char* name)
{
    FILE* fp=NULL;
    ext2_dir_entry dir;
    ext2_inode node;
    time_t now;
    char str;
    int i;
    while (fp==NULL)
    {
        fp=fopen(PATH,"r+");
    }
    while (1)
    {
        for (i=0;i<(current->i_size/32);i++)
        {
            fseek(fp,dir_entry_position(i*32,current->i_block),SEEK_SET);
            fread(&dir,sizeof(ext2_dir_entry),1,fp);
            if (!strcmp(dir.name,name))
            {
                if (dir.file_type==1)
                {
                    fseek(fp,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
                    fread(&node,sizeof(ext2_inode),1,fp);

                    break;
                }
```

```
            }
        }
        if (i<current->i_size/32)
        {
            break;
        }
        //Create(1,current,name);
        printf("There is not such a file,creaate it first.\n");
        return 0;
    }
    str=getch();
    while (str!=27)
    {
        printf("%c",str);
        if (!(node.i_size%512))
        {
            add_block(&node,node.i_size/512,FindBlock());
            node.i_blocks+=1;
        }
        fseek(fp,dir_entry_position(node.i_size,node.i_block),SEEK_SET);
        fwrite(&str,sizeof(char),1,fp);
        node.i_size+=sizeof(char);
        if (str==0x0d) printf("%c",0x0a);
        str=getch();
        if (str==27) break;
    }
    time(&now);
    node.i_mtime=now;
    node.i_atime=now;
    fseek(fp,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
    fwrite(&node,sizeof(ext2_inode),1,fp);
    fclose(fp);
    printf("\n");
    return 0;
}

void Ls(ext2_inode* current)
{
    ext2_dir_entry dir;
    int i,j;
    char timestr[150];
    ext2_inode node;
    f=fopen(PATH,"r+");
    printf("Type\tFileName\tCreateTime\tLastAccessTime\tModifyTime\n");
    for (int i=0;i<(current->i_size/32);i++)
    {
        fseek(f,dir_entry_position(i*32,current->i_block),SEEK_SET);
        fread(&dir,sizeof(ext2_dir_entry),1,f);
        fseek(f,3*blocksiz+dir.inode*sizeof(ext2_inode),SEEK_SET);
        fread(&node,sizeof(ext2_inode),1,f);
        strcpy(timestr," ");
        strcat(timestr,asctime(localtime(&node.i_ctime)));
```

```
        strcat(timestr,asctime(localtime(&node.i_atime)));
        strcat(timestr,asctime(localtime(&node.i_mtime)));
        for (j=0;j<strlen(timestr)-1;j++)
        {
            if (timestr[j]=='\n') timestr[j]='\t';
        }
        if (dir.file_type==1) printf("File %s\t%s",dir.name,timestr);
        else printf("Directory %s\t%s",dir.name,timestr);

    }
    fclose(f);
}

int initialize(ext2_inode* cu)
{
    f=fopen(PATH,"r+");
    fseek(f,3*blocksiz,0);
    fread(cu,sizeof(ext2_inode),1,f);
    fclose(f);
    return 0;
}

int Password()
{
    char psw[16],ch[0],new[16];
    printf("Input old password:\n");
    scanf("%s",psw);
    if (strcmp(psw,group_desc.psw)!=0)
    {
        printf("Password erreo.\n");
        return 1;
    }
    while (1)
    {
        printf("Input new password:\n");
        scanf("%s",new);
        while (1)
        {
            printf("Modify passworld?[Y/N]");
            scanf("%s",ch);
            if (ch[0]=='N'||ch[0]=='n')
            {
                printf("Cancel.\n");
                return 1;
            }
            else if(ch[0]=='Y'||ch[0]=='y')
            {
                strcpy(group_desc.psw,new);
                f=fopen(PATH,"r+");
                fseek(f,0,0);
                fwrite(&group_desc,sizeof(ext2_group_desc),1,f);
                fclose(f);
```

```
                return 0;
            }
            else printf("Meaningless command.\n");
        }
    }

}
//登陆 无bug
int login()
{
    char psw[16];
    printf("Input password(init:123):");
    scanf("%s",psw);
    return strcmp(group_desc.psw,psw);
}

void exitdisplay()
{
    printf("bye~\n");
    return ;
}

int initfs(ext2_inode* cu)
{
    f=fopen(PATH,"r+");
    if (f==NULL)
    {
        char ch;
        int i;
        printf("File system:Not Found.Create one?\n[Y/N]");
        i=1;
        while(i)
        {
            scanf("%c",&ch);
            switch ((ch))
            {
            case 'Y':
            case 'y':
                if (format()!=0) return 1;
                f=fopen(PATH,"r");
                i=0;
                break;
            case 'N':
            case 'n':
                exitdisplay();
                return 1;
            default:
                printf("Meaningless command.\n");
                break;
            }
        }
    }
```

```
    fseek(f,0,SEEK_SET);
    fread(&group_desc,sizeof(ext2_group_desc),1,f);
    fseek(f,3*blocksiz,SEEK_SET);
    fread(&inode,sizeof(ext2_inode),1,f);
    fclose(f);
    initialize(cu);
    return 0;
}

//获得当前目录名
void getstring(char* cs,ext2_inode node)
{
    ext2_inode current=node;
    int i,j;
    ext2_dir_entry dir;
    f=fopen(PATH,"r+");
    Open(&current,"..");
    for (i=0;i<node.i_size/32;i++)
    {
        fseek(f,dir_entry_position(i*32,node.i_block),SEEK_SET);
        fread(&dir,sizeof(ext2_dir_entry),1,f);
        if (!strcmp(dir.name,"."))
        {
            j=dir.inode;
            break;
        }
    }
    for (i=0;i<current.i_size/32;i++)
    {
        fseek(f,dir_entry_position(i*32,current.i_block),SEEK_SET);
        fread(&dir,sizeof(ext2_dir_entry),1,f);
        if (dir.inode==j)
        {
            strcpy(cs,dir.name);
            return;
        }
    }
}

int Delet(int type,ext2_inode* current,char* name)
{
    FILE* fout=NULL;
    int i,j,k,t,flag;
    int Blocation2,Blocation3;
    int node_location,dir_entry_location,block_location;
    int block_location2,block_location3;
    ext2_inode cinode;
    ext2_dir_entry bentry,centry,dentry;
    dentry.inode=0;
    dentry.rec_len=sizeof(ext2_dir_entry);
    dentry.name_len=0;
    dentry.file_type=0;
```

```
    strcpy(dentry.name,"");
    dentry.dir_pad=0;
    fout=fopen(PATH,"r+");
    t=(int)(current->i_size/dirsiz);
    flag=0;
    fseek(fout,(data_begin_block+current->i_block[0])*blocksiz,SEEK_SET);
    fread(&bentry,dirsiz,1,fout);
    //printf("When delete,inode:%d\n",bentry.inode);
    for (i=0;i<t;i++)
    {
        dir_entry_location=dir_entry_position(i*dirsiz,current->i_block);
        fseek(fout,dir_entry_location,SEEK_SET);
        fread(&centry,sizeof(ext2_dir_entry),1,fout);
        if ((strcmp(centry.name,name))==0&&(centry.file_type==type))
        {
            flag=1;
            j=i;
            break;
        }

    }
    if (flag)
    {
        node_location=centry.inode;
        fseek(fout,3*blocksiz+node_location*sizeof(ext2_inode),SEEK_SET);
        fread(&cinode,sizeof(ext2_inode),1,fout);
        block_location=cinode.i_block[0];
        if (type==2) //删除目录
        {
            if (cinode.i_size>2*dirsiz)
            {
                ext2_inode next=cinode;
                ext2_dir_entry subdir;
                for (int m=2;m<cinode.i_size/dirsiz;m++)
                {

fseek(fout,(data_begin_block+cinode.i_block[0])*blocksiz+m*dirsiz,SEEK_SET);
                    fwrite(&subdir,dirsiz,1,fout);

                    Delet(subdir.file_type,&next,subdir.name);

                }
                // printf("The folder is not empty.\n");
                // return 1;

            }
            //else //目录已空
            //{
                DelBlock(block_location);
                DelInode(node_location);
                dir_entry_location=dir_entry_position(current->i_size-
dirsiz,current->i_block);
```

```
                fseek(fout,dir_entry_location,SEEK_SET);
                fread(&centry,dirsiz,1,fout);
                fseek(fout,dir_entry_location,SEEK_SET);
                fwrite(&dentry,dirsiz,1,fout);
                dir_entry_location-=data_begin_block*blocksiz;
                if (dir_entry_location%blocksiz==0)
                {
                    DelBlock((int)(dir_entry_location/blocksiz));
                    current->i_blocks--;
                    if (current->i_blocks==6) DelBlock(current->i_block[6]);
                    else if (current->i_blocks==(blocksiz/sizeof(int)+6))
                    {
                        int a;
                        fseek(fout,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz,SEEK_SET);
                        fread(&a,sizeof(int),1,fout);
                        DelBlock(a);
                        DelBlock(current->i_block[7]);
                    }
                    else if (!((current->i_blocks-6-
blocksiz/sizeof(int))%(blocksiz/sizeof(int))))
                    {
                        int a;
                        fseek(fout,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz+
                            ((current->i_blocks-6-
blocksiz/sizeof(int))/(blocksiz/sizeof(int))),SEEK_SET);
                        fread(&a,sizeof(int),1,fout);
                        DelBlock(a);
                    }
                }
                current->i_size-=dirsiz;
                //printf("%d\n",current->i_size);
                if (j*dirsiz<current->i_size-dirsiz)
                {
                    dir_entry_location=dir_entry_position(j*dirsiz,current-
>i_block);
                    fseek(fout,dir_entry_location,SEEK_SET);
                    fwrite(&centry,dirsiz,1,fout);
                }
                printf("%s has been deleted.\n",name);
            //}
        }
        else //删除文件
        {
            for (i=0;i<6;i++)
            {
                if (cinode.i_blocks==0) break;
                block_location=cinode.i_block[i];
                DelBlock(block_location);
                cinode.i_blocks--;
            }
```

```
            if (cinode.i_blocks>0)
            {
                block_location=cinode.i_block[6];
                fseek(fout,(data_begin_block+block_location)*blocksiz,SEEK_SET);
                for (i=0;i<blocksiz/sizeof(int);i++)
                {
                    if (cinode.i_blocks==0) break;
                    fread(&Blocation2,sizeof(int),1,fout);
                    DelBlock(Blocation2);
                    cinode.i_blocks--;
                }
                DelBlock(block_location);
            }
            if (cinode.i_blocks>0)
            {
                block_location=cinode.i_block[7];
                for (i=0;i<blocksiz/sizeof(int);i++)
                {

fseek(fout,(data_begin_block+block_location)*blocksiz+i*sizeof(int),SEEK_SET);
                    fread(&Blocation2,sizeof(int),1,fout);
                    fseek(fout,(data_begin_block+Blocation2)*blocksiz,SEEK_SET);
                    for (k=0;k<blocksiz/sizeof(int);k++)
                    {
                        if (cinode.i_blocks==0) break;
                        fread(&Blocation3,sizeof(int),1,fout);
                        DelBlock(Blocation3);
                        cinode.i_blocks--;
                    }
                    DelBlock(Blocation2);
                }
                DelBlock(block_location);
            }
            DelInode(node_location);
            dir_entry_location=dir_entry_position(current->i_size-dirsiz,current-
>i_block);
            fseek(fout,dir_entry_location,SEEK_SET);
            fread(&centry,dirsiz,1,fout);
            fseek(fout,dir_entry_location,SEEK_SET);
            fwrite(&dentry,dirsiz,1,fout);
            dir_entry_location-=data_begin_block*blocksiz;
            if (dir_entry_location%blocksiz==0)
            {
                DelBlock((int)(dir_entry_location/blocksiz));
                current->i_blocks--;
                if (current->i_blocks==6) DelBlock(current->i_block[6]);
                else if (current->i_blocks==(blocksiz/sizeof(int)+6))
                {
                    int a;
                    fseek(fout,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz,SEEK_SET);
                    fread(&a,sizeof(int),1,fout);
```

```
                    DelBlock(a);
                    DelBlock(current->i_block[7]);
                }
                else if (!((current->i_blocks-6-
blocksiz/sizeof(int))%(blocksiz/sizeof(int))))
                {
                    int a;
                    fseek(fout,data_begin_block*blocksiz+current-
>i_block[7]*blocksiz+
                        ((current->i_blocks-6-
blocksiz/sizeof(int))/(blocksiz/sizeof(int))),SEEK_SET);
                    fread(&a,sizeof(int),1,fout);
                    DelBlock(a);
                }
            }
            current->i_size-=dirsiz;
            if (j*dirsiz<current->i_size)
            {
                dir_entry_location=dir_entry_position(j*dirsiz,current->i_block);
                fseek(fout,dir_entry_location,SEEK_SET);
                fwrite(&centry,dirsiz,1,fout);
            }

        }
        fseek(fout,(data_begin_block+current->i_block[0])*blocksiz,SEEK_SET);
        fread(&bentry,dirsiz,1,fout);
        //printf("When delete,inode:%d\n",bentry.inode);
        fseek(fout,3*blocksiz+(bentry.inode)*sizeof(ext2_inode),SEEK_SET);
        //printf("%d\n",current->i_size);
        fwrite(current,sizeof(ext2_inode),1,fout);
    }
    else
    {
        fclose(fout);
        return 1;
    }
    fclose(fout);
    return 0;
}

void shellloop(ext2_inode currentdir)
{
    char command[10],var1[10],var2[100],var3[128],path[100];
    ext2_inode temp;
    int i,j;
    char currentsting[20];
    char ctable[12][10]={"create","delete","cd","close","read","write",
    "password","format","exit","login","logout","ls"};
    while (1)
    {
        getstring(currentsting,currentdir);
        printf("\n%s=>#",currentsting);
```

```
        scanf("%s",command);
        for (i=0;i<12;i++)
        {
            if (!strcmp(command,ctable[i])) break;
        }
            if (i==0||i==1)
            {
                scanf("%s",var1);
                scanf("%s",var2);
                if (var1[0]=='f') j=1;
                else if (var1[0]=='d') j=2;
                else
                {
                    printf("The first variant must be [f/d]");
                    continue;
                }
                if (i==0)
                {
                    if (Create(j,&currentdir,var2)==1) printf("%s faild to
create.\n",var2);
                    else printf("%s created.\n",var2);
                }
                else
                {
                    if (Delet(j,&currentdir,var2)==1)
                    {
                        printf("%s failed to delete.\n",var2);
                    }
                    else
                    {
                        printf("%s deleted.\n",var2);
                    }
                }
            }
            else if (i==2)
            {
                scanf("%s",var2);
                i=0;j=0;
                temp=currentdir;
                while ((1))
                {
                    path[i]=var2[j];
                    if (path[i]=='/')
                    {
                        if (j==0) initialize(&currentdir);
                        else if (i==0)
                        {
                            printf("Path input error.\n");
                            break;
                        }
                        else
                        {
```

```
                    path[i]='\0';
                    if (Open(&currentdir,path)==1)
                    {
                        printf("Path input error.\n");
                        currentdir=temp;
                    }
                }
                i=0;
            }
            else if (path[i]=='\0')
            {
                if (i==0) break;
                if (Open(&currentdir,path)==1)
                {
                    printf("Path input error.\n");
                    currentdir=temp;
                }
                break;
            }
            else i++;
            j++;
        }
    }
    else if (i==3)
    {
        scanf("%d",&i);
        for (j=0;j<i;j++)
        {
            if (Close(&currentdir)==1)
            {
                printf("Warning:%d is too large.\n",i);
                break;
            }
        }
    }
    else if (i==4)
    {
        scanf("%s",var2);
        if (Read(&currentdir,var2)==1)
        {
            printf("Failed:can't read.\n");
        }
    }
    else if (i==5)
    {
        scanf("%s",var2);
        if (Write(&currentdir,var2)==1)
        {
            printf("Failed:can't write.\n");
        }
    }
    else if (i==6)
```

```
            {
                Password();
            }
            else if (i==7)
            {
                while (1)
                {
                    printf("Do you want to format the filesystem?\nIt will be
dangerous.\n");
                    printf("[Y/N]");
                    scanf("%s",var1);
                    if (var1[0]=='N'||var1[0]=='n') break;
                    else if (var1[0]=='Y'||var1[0]=='y')
                    {
                        format();
                        break;
                    }
                    else printf("Input [Y/N]");
                }

            }
            else if (i==8)
            {
                while (1)
                {
                    printf("Do you want to exit from filesystem?[Y/N]");
                    scanf("%s",var2);
                    if (var2[0]=='N'||var2[0]=='n') break;
                    else if (var2[0]=='Y'||var2[0]=='y') return;
                    else printf("Input [Y/N]");
                }

            }
            else if (i==9)
            {
                printf("Failed:You havn't logged out.\n");
            }
            else if (i==10)
            {
                while (i)
                {
                    printf("Do you want to log out?[Y/N]");
                    scanf("%s",var1);
                    if (var1[0]=='N'||var1[0]=='n') break;
                    else if (var1[0]=='Y'||var1[0]=='y')
                    {
                        initialize(&currentdir);
                        while (1)
                        {
                            printf("$$$$=>#");
                            scanf("%s",var2);
                            if (strcmp(var2,"login")==0)
```

```
                            {
                                if (login()==0)
                                {
                                    i=0;
                                    break;
                                }
                            }
                            else if (strcmp(var2,"exit")==0) return;
                            else printf("Inpur [Y/N]");
                        }

                    }
                }

            }
            else if (i==11)
            {
                Ls(&currentdir);
            }
            else printf("Failed:invalid command.\n");
    }
}

int main()
{
    ext2_inode cu;
    printf("Welcome!\n");
    if (initfs(&cu)==1) return 0;
    if (login()!=0)
    {
        printf("Incorrect password.It will terminate right away.");
        exitdisplay();
        return 0;
    }
    shellloop(cu);
    exitdisplay();
    return 0;
}
```

3.7.2 附件 2 Readme

遇到的问题：

1. 类 ext2 文件系统的基本功能对应的命令的作用各不相同实现的细节较为繁琐

2. 用文件模拟磁盘进行操作时，是一个二进制文件，像磁盘一样对程序员不可见，因此当输出结果不对时，不方便通过打印的方式定位错误的原因

3. 各模块之间调用关系嵌套，给调试带来困难

尝试解决的对策：

1. 用模块化设计的方式，逐个实现要完成的功能，经过调试确保无误。例如，

先实现简单的 format(),login(),logout()，然后根据系统测试的流程，依此实现 ls，cd，create，write，read，delete 等功能。

2. 为处理多个方法调用嵌套为 debug 带来的阻碍，可以将不同的功能先在不同的源代码中实现，再集成

3. 分别调试不同功能的时候，就可以打断点调试查错

4. 仔细思考磁盘逻辑分区的方法和细节，做好笔记。模拟磁盘不同分区的读写通过文件指针的移动来实现，数学计算比较复杂，尤其是涉及混合索引的不分要仔细检查。

命令功能：

1. format 初始化磁盘，建立组描述符和根目录，为其分配索引节点和磁盘块

2. login 用户登录

3. ls 列表，获得当前目录所有项的 list

4. create 创建文件或目录，分配索引节点和磁盘块，命名目录项

5. delete 删除文件或目录，释放索引节点和磁盘块，清除目录项

6. cd 进入当前目录下的一个目录，并且当前目录指针指向其

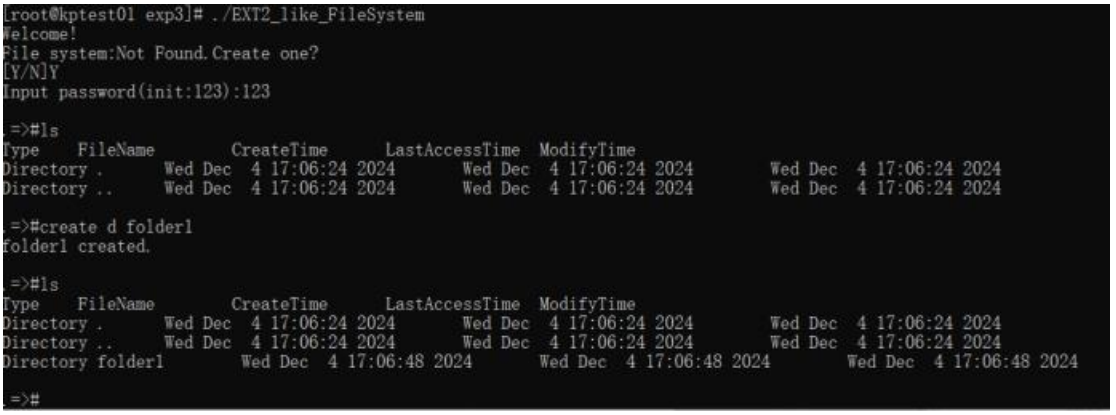
7. write 向文件写入内容（需合法）

8. read 读出文件内容（需合法）

9. password 修改登录密码

10.exit 退出终端

测试结果：



```
=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024
Directory ..      Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024
Directory folder1       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024

=>#cd folder1

folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024
Directory ..      Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024

folder1=>#create f file1
file1 created.

folder1=>#create f file2
file2 created.

folder1=>#write file1

Hello world!

folder1=>#
```

```
folder1=>#write file1

Hello world!

folder1=>#write file2

Good evening!


folder1=>#read file1

Hello world!

folder1=>#read file2

Good evening!


folder1=>#delete f file1
file1 deleted.

folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024
Directory ..      Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024
File file2        Wed Dec  4 17:07:47 2024       Wed Dec  4 17:08:55 2024       Wed Dec  4 17:08:47 2024

folder1=>#
```

```
folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024
Directory ..      Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024
File file2        Wed Dec  4 17:07:47 2024       Wed Dec  4 17:08:55 2024       Wed Dec  4 17:08:47 2024

folder1=>#delete f file2
file2 deleted.

folder1=>#ls
Type    FileName        CreateTime      LastAccessTime  ModifyTime
Directory .       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024       Wed Dec  4 17:06:48 2024
Directory ..      Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024       Wed Dec  4 17:06:24 2024

folder1=>#cd ..

=>#logout
Do you want to log out?[Y/N]Y
$$$$=>#exit
bye
[root@kptest01 exp3]#
```