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

1.1.1 进程相关编程 实验目的

（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 .实验内容

（0）开通华为云并创建 OpenEuler 实验环境。

（1）熟悉操作命令、编辑、编译、运行程序。完成图 1-1 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。

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

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d",pid); /* A */
       printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }

    return 0;
}
```

（2）扩展上图的程序：
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；

1.1.3 实验思想

（1）通过使用华为云平台开通云服务器，初步了解 OpenEuler 操作系统，了解并实践基本的命令操作。

（2）通过完成进程， 线程，自旋锁的实验，理解自旋锁的运行原理以及进程和线程的异同。

（2）理解并掌握相关进程线程相关库函数的基本应用。

1.1.4 实验步骤

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：编写并多次运行图中代码。

步骤二：删去图 1-1 代码中的 wait()函数并多次运行程序，分析运行结果。

步骤三：修改图 1-1 中代码，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：修改图 1-1 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。

1.1.5 测试数据设计

对于实验步骤一，无需设计测试数据，直接运行程序即可；

实验步骤二同理，删去 wait()函数后运行程序即可。

对于实验步骤三，我设计了全局变量 global，并将其初始化为 2004，在子进程中将其修改为 24，在父进程中将其修改为 88；

对于实验步骤四，在返回语句之前将全局变量 global 修改为 2004+pid；

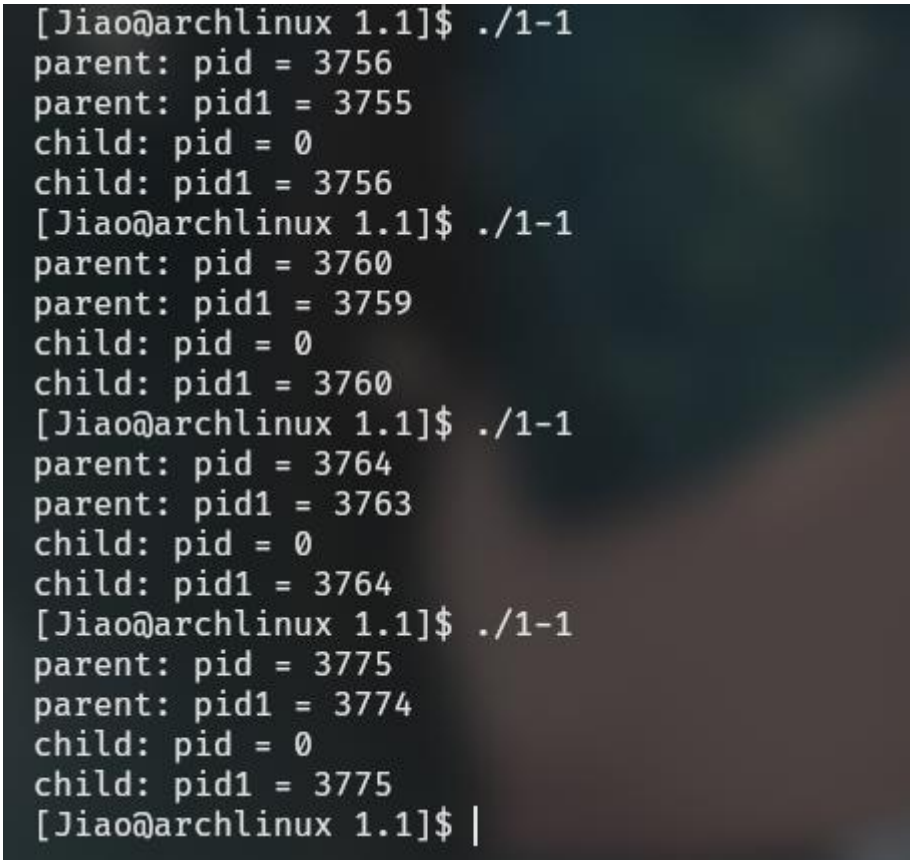
对于实验步骤 5，我分别编写了 system_call.c 和 execl.c 并将它们编译生成可执行文件 system_call 和 execl, 并在子进程中分别用 system ()和 execl()调用执行这两个文件。

1.1.6 程序运行初值及运行结果分析

步骤 1：

直接执行原始代码即可。

以下为运行结果：



运行结果分析：

pid_t 类型的变量是用来记录 process identity 的量，在父进程中调用 fork()函数时，会创建一个子进程，在父进程中返回子进程的 pid，在子进程中返回 0；在程序中调用 getpid()函数时，会返回当前进程的 pid；由于子进程在父进程之后才创建，因此子进程 pid 会比父进程大。

分析程序可知，在父进程中，‘pid’是子进程的 pid，‘pid1’是父进程自己的 pid；

在子进程中，‘pid’为 0，pid1 为子进程自己的 pid。

由此可知程序运行正确，符合预期。

步骤 2：删除父进程中的 wait(NULL); 后再次运行程序。

以下为运行结果：

```
[Jiao@archlinux 1.1]$ gcc 1-1-wait.c -o 1-1-nowait
[Jiao@archlinux 1.1]$ ./1-1-nowait
parent: pid = 3909
parent: pid1 = 3908
child: pid = 0
child: pid1 = 3909
[Jiao@archlinux 1.1]$ ./1-1-nowait
parent: pid = 3913
parent: pid1 = 3912
child: pid = 0
child: pid1 = 3913
[Jiao@archlinux 1.1]$ ./1-1-nowait
parent: pid = 3918
parent: pid1 = 3917
child: pid = 0
child: pid1 = 3918
[Jiao@archlinux 1.1]$ ./1-1-nowait
parent: pid = 3922
parent: pid1 = 3921
child: pid = 0
child: pid1 = 3922
[Jiao@archlinux 1.1]$ ./1-1-nowait
parent: pid = 3926
parent: pid1 = 3925
child: pid = 0
child: pid1 = 3926
[Jiao@archlinux 1.1]$
```

运行结果分析：

wait()函数的作用是如果父进程运行比子进程快，当父进程运行到 wait()函数语句时，会暂时停止运行，等待一个子进程运行结束之后再继续运行（如果函数有参数，那么参数会承接 terminated 的子进程 pid），这个函数可以保证父进程在子进程之后才被 terminate，防止因父进程先执行结束导致子进程变为 zombie 进程。

比对之后可以发现运行结果于步骤一几乎一致，这是因为 wait(NULL); 是父进程执行的最后一条语句，并不会影响输出结果，不过确实可能会导致父进程在子进程之前结束，只不过不会影响到输出结果。（输出顺序与 CPU 调度有关，并非 wait 导致）。

步骤 3：父子进程分别对一全局变量进行不同操作，打印不同进程中全局变量的值和地址。

以下为运行结果：

```
[Jiao@archlinux 1.1]$ ./1-2
parent: pid = 4065
parent: pid1 = 4064
child: pid = 0
child: pid1 = 4065
child's global = 24 , add= -1185304520
parent's global = 88 , add =-1185304520
[Jiao@archlinux 1.1]$ ./1-2
parent: pid = 4074
parent: pid1 = 4073
child: pid = 0
child: pid1 = 4074
child's global = 24 , add= 1318506552
parent's global = 88 , add =1318506552
[Jiao@archlinux 1.1]$ ./1-2
parent: pid = 4078
parent: pid1 = 4077
child: pid = 0
child: pid1 = 4078
child's global = 24 , add= 51843128
parent's global = 88 , add =51843128
[Jiao@archlinux 1.1]$ ./1-2
parent: pid = 4082
parent: pid1 = 4081
child: pid = 0
child: pid1 = 4082
child's global = 24 , add= -929451976
parent's global = 88 , add =-929451976
[Jiao@archlinux 1.1]$
```

运行结果分析：

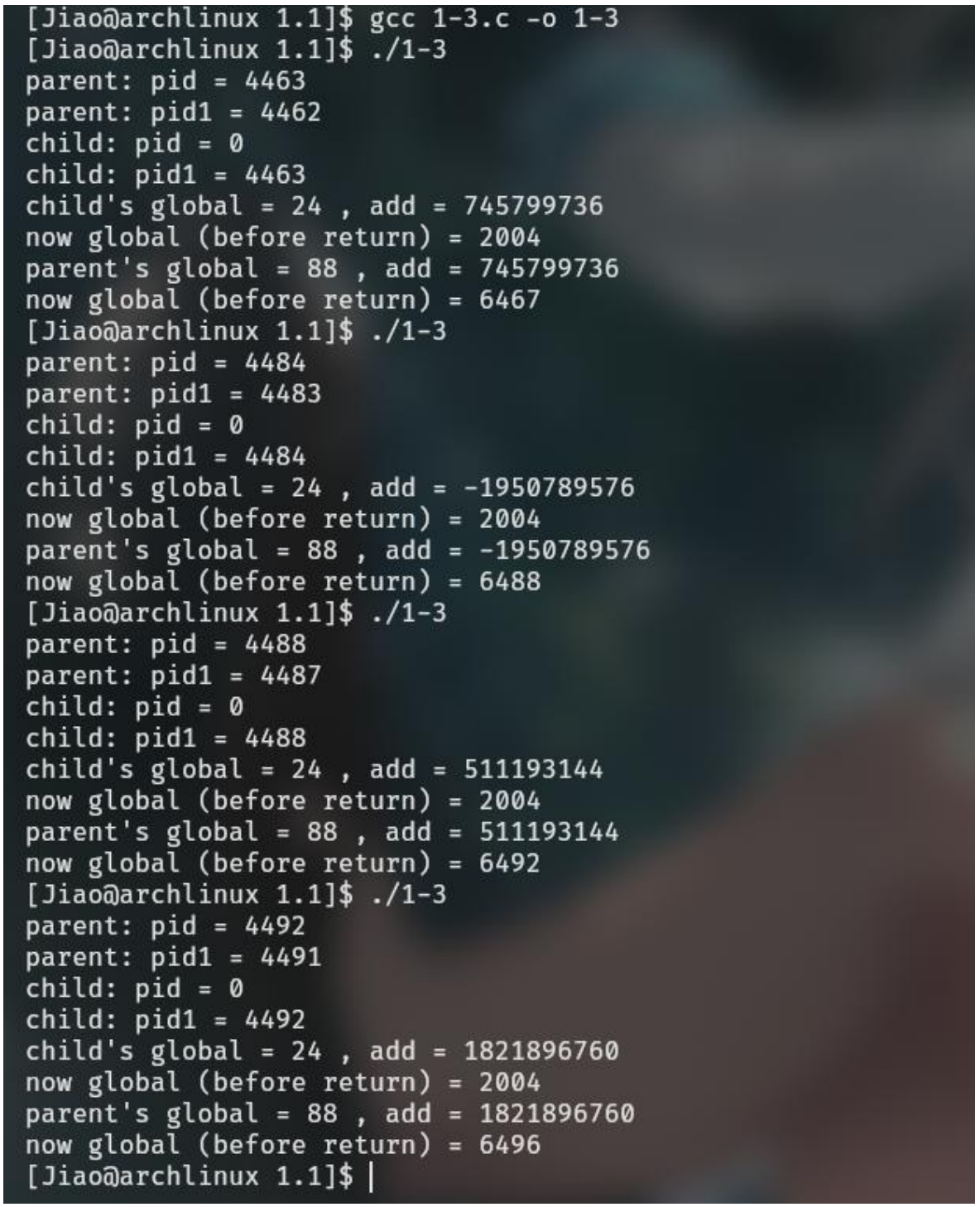
子进程在创建时，会形成一个父进程完全的 duplicate，但是子进程中的变量与父进程却是相互独立的，因此在父进程和子进程中进行修改，得到的是不同的结果；

子进程完全 duplicate 父进程，所以打印出的地址与父进程中也是一致的，但是这并不是说在同一个地址中存储了两个不同的数据，为了节省资源，子进程并不会在创建时立刻实际上就生成一份父进程的拷贝，而是暂时和父进程公用数据，直到数据被修改，子进程和父进程中数据不一致时，才会真的新开空间存储数据，利用了 copy-on-write 技术。

同时，地址即便相同，也是操作系统提供给用户的“假象”，这实际上是一个虚拟地址，即使子进程和父进程对此变量的虚拟地址一致，他们也会被映射到不同的物理地址。

步骤 4：在步骤 3 的基础上，在 return 之前打印全局变量 value 的值和地址。

以下是运行结果：

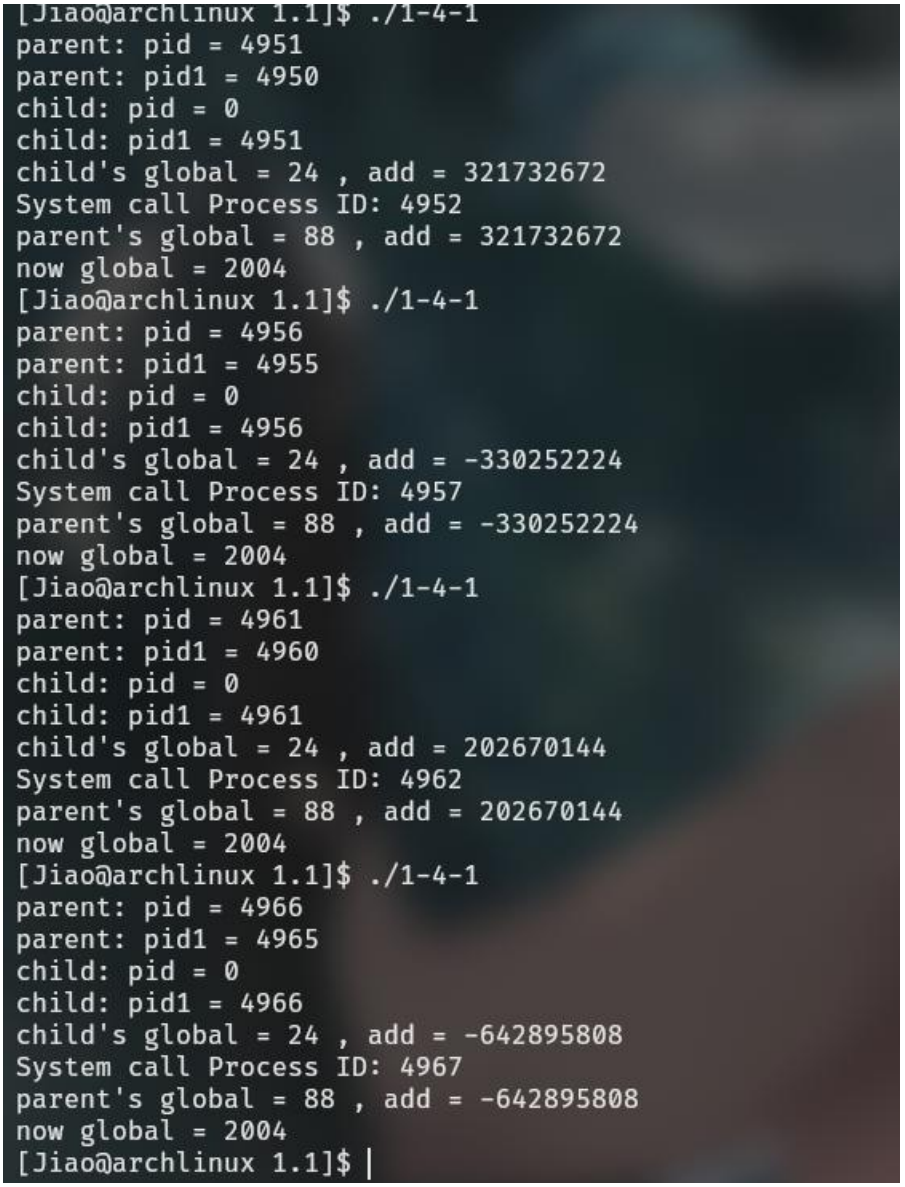


结果分析：

子进程和父进程在结束之前，都会执行这个语句，在此处输出的值就可以很明显的看出，两个进程中的变量是相互独立的，进一步说明了**步骤三中结果分析**的正确性。

步骤 5：在子进程中调用 system()与 exec 族函数

以下为 system()调用的运行结果：

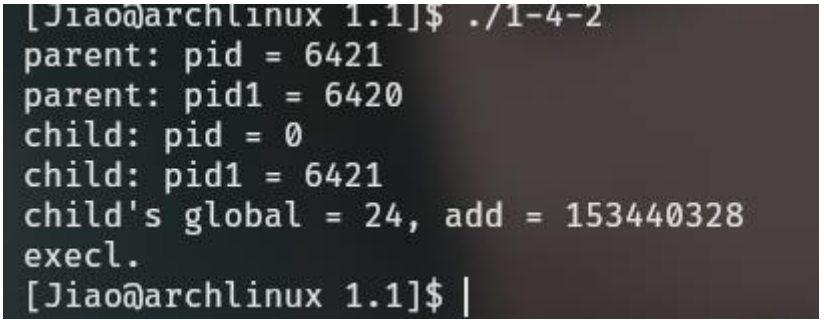


结果分析：

这里可以发现，system_call 程序在运行时输出的 pid 比子进程大 pid 要大 1，system_call 运行结束后，该进程剩余程序仍然会继续执行。

这是由于，当 system()被调用时，当前进程会立刻创建一个新的进程，并暂时停止运行当前进程而先运行 system()创建的新进程，当新进程运行结束后， 会返回原来的进程继续运行。

以下为 exec 族函数的运行结果：



结果分析：

这里可以发现，execl()运行结束后，子进程立即停止运行。

这是由于，execl()被调用时，并不是创建新的进程，而是把当前程序内容替换为需要执行的程序，所以 pid 没有增加，运行结束后子进程之后的部分也不再运行了。

1.2.1 线程相关编程实验 实验目的

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。

1.2.2 实验内容

（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 实验思想

通过创建两个线程，它们分别对一个共享的变量进行多次循环操作，并观察在多次运行实验时可能出现的不同结果。在观察到结果不稳定的情况下，引入互斥和同步机制来确保线程间的正确协同操作。

1.2.4 实验步骤

步骤 1：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 2：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 3：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。

1.2.5 测试数据设计

定义全局变量 global，初始化为 2024；

定义两个线程，在两个线程中，对 global 执行不同的操作：在线程 1 中，循环 5001 次，每次循环对 global 加 1；在第二个线程中，每次令 global 减 2.

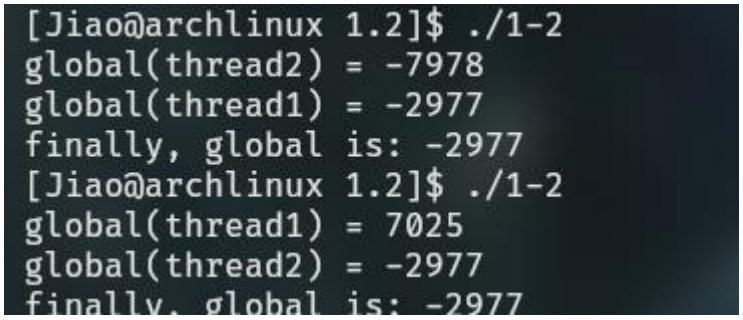
步骤三中测试数据与线程实验中的类似，也定义了 system_call 可执行文件并在线程中进行调用。

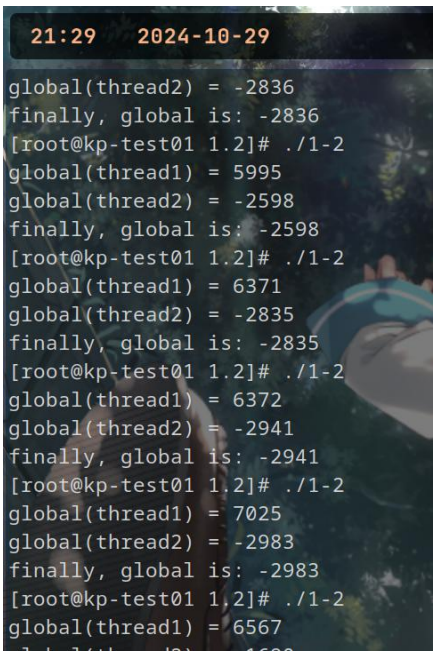
1.1.6 程序运行初值及运行结果分析

步骤 1：

两个线程分别对 global 执行各自的操作。

以下为运行结果：





结果分析：

可以发现，运行结果并不稳定，这是由于两个进程同时对一个变量进行不原子的操作时，会出现竞态，由于 CPU 调度的不确定性导致输出结果变化不稳定。

步骤 2：

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。

以下为运行结果：

```
[Jiao@archlinux 1.2]$ ./1-2-3
global(thread1) = -1429
global(thread2) = -2977
finally, global is: -2977
[Jiao@archlinux 1.2]$ ./1-2-3
global(thread1) = 3593
global(thread2) = -2977
finally, global is: -2977
[Jiao@archlinux 1.2]$ ./1-2-3
global(thread1) = -2159
global(thread2) = -2977
finally, global is: -2977
[Jiao@archlinux 1.2]$ ./1-2-3
global(thread1) = -481
global(thread2) = -2977
finally, global is: -2977
[Jiao@archlinux 1.2]$
```

结果分析：

在加入互斥锁后，当一个线程在对共享变量进行操作时，另一个线程无法对共享变量进行操作，这样可以让两个线程“有序”地执行，所以最终会得到一个稳定地结果。（但是由于 CPU 调度地不确定性，只能保证线程每一次循环都在无干扰地情况下进行而不能保证两个线程运行的顺序，所以过程量可能不同而最终结果可以保持稳定一致）

步骤 3：

在两个线程中分别调用 system(“./system_call”)，运行结果如下：

```
[Jiao@archlinux 1.2]$ ./1-2-4
System call Process ID: 7755
System call Process ID: 7754
thread1 pid=7751,tid=7752
thread2 pid=7751,tid=7753
[Jiao@archlinux 1.2]$ ./1-2-4
System call Process ID: 7761
System call Process ID: 7762
thread1 pid=7758,tid=7759
thread2 pid=7758,tid=7760
[Jiao@archlinux 1.2]$ ./1-2-4
System call Process ID: 7769
System call Process ID: 7768
thread1 pid=7765,tid=7766
thread2 pid=7765,tid=7767
[Jiao@archlinux 1.2]$ ./1-2-4
System call Process ID: 7791
System call Process ID: 7792
thread1 pid=7788,tid=7789
thread2 pid=7788,tid=7790
[Jiao@archlinux 1.2]$
```

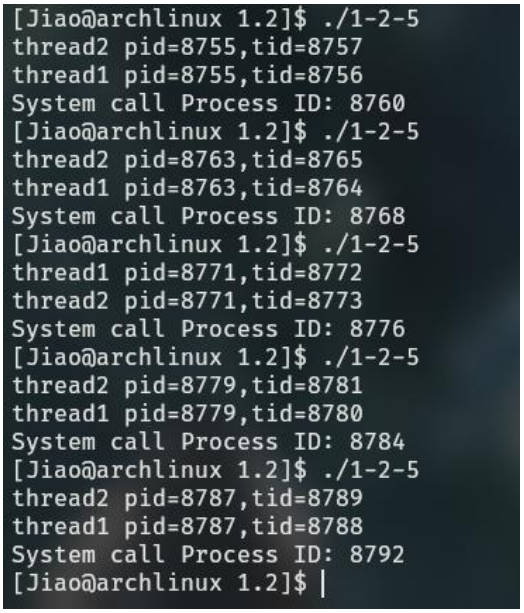
结果分析：

可以发现，两线程输出的 pid 是一样的，可以说明两个线程时属于同一个进程的；

两个线程输出的 tid 是不同的，相差 1，说明他们是同一个进程先后创建的两个线程；

system_call 中显示的 pid 是不同的，可以说明，system()是会创建新的进程来执行相应程序的。

在两个线程中调用 execl()函数，运行结果如下：



结果分析：

线程输出的 pid 和 tid 结果与 system()的一致，在此不做分析。

其中一个线程调用 execl()后，由于当前进程被替换为 execl()要执行的部分，所以两个线程也被替换了，所以 execl()只被执行一次。

1.3.1 自旋锁实验 实验目的

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。

1.3.2 实验内容

（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；

1.3.3 实验思想

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。它的工作原理可以简单地概括为以下几个步骤：

（1）初始化锁： 自旋锁的开始是一个共享的标志变量（flag），最初为未锁定状态（0）。这个标志变量用于表示资源是否已被其他线程占用。

（2）获取锁： 当一个线程尝试获取锁时，它会循环检查标志变量的状态。如果发现标志变量是未锁定状态（0），那么该线程将通过原子操作将标志变量设置为锁定状态（1），从而成功获取锁。如果标志变量已经是锁定状态，线程会一直在循环中等待，直到标志变量变为未锁定状态为止。

（3）释放锁： 当持有锁的线程完成对共享资源的操作后，它会通过原子操作将标志变量设置回未锁定状态（0），从而释放锁，允许其他等待的线程尝试获取锁。

自旋锁的工作原理中关键的部分在于“自旋”这一概念，即等待获取锁的线程会循环忙等待，不断检查标志变量的状态，直到能够成功获取锁。这种方式在锁的占用时间很短的情况下可以减少线程切换的开销，提高程序性能。

1.3.4 实验步骤

步骤 1：根据实验内容要求，编写模拟自旋锁程序代码 spinlock.c

步骤 2：运行补充后的文件

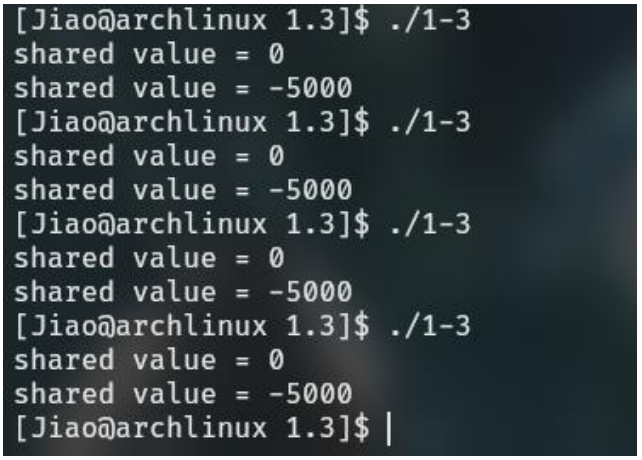
1.3.5 测试数据设计

定义共享变量初始值为 0，在第一个线程中循环 5000 次，每次对共享变量+1；在第二个线程中循环 5000 次，每次对共享变量-2；

1.3.6 程序运行初值及运行结果分析

初始时将 shared_value 设置为 0，两个线程分别对其进行各自的操作，输出最终结果。

运行结果如下：



结果分析：

由于每次进行运算时，都使用了自旋锁对共享变量进行保护，因此输出的结果是正确而稳定的。

1.7 实验总结

1.7.1 实验中的问题与解决过程

遇到的问题：

在使用 exec 族函数运行编译生成的 system_call 时，出现了提示权限不足的问题，在 chmod 之后依然提示权限不足。

解决过程：

在 execl()中传入”/bin/sudo” ,“sudo”，于是可以成功调用外部可执行程序。

1.7.2 实验收获

通过本次实验，让我对进程，线程，自旋锁的实现原理和 CPU 的调度方式有了更深的理解，理解了子进程创建时的操作原理，copy-on-write ，虚拟地址等技术，理解了 pid，tid 之间的关系，OS 对两者的不同调度方式。

通过本次实验，我还熟悉了 fork，system，execl 等函数的使用，同时加深了对多进程，多线程的理解。

1.7.3 意见与建议

可以在给出的代码中多打几行注释，简要说明某函数的作用；

在较长的代码中，解释各个变量代表什么，有怎样的数据结构，以便理解。

1.8 附件

1.8.1 附件 1 程序

进程实验：
代码 1：

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

int main(){

    pid_t pid,pid1;

    pid=fork();
    if(pid<0){
        fprintf(stderr,"Fork Failed");
        return 1;
    }
    else if(pid==0){
        pid1=getpid();
        printf("child: pid = %d \n",pid);
        printf("child: pid1 = %d \n",pid1);

    }

    else{
        pid1=getpid();
        printf("parent: pid = %d \n",pid);
        printf("parent: pid1 = %d \n",pid1);
        wait(NULL);


    }
    return 0;
```

```
}
```

代码 2：

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

int main(){

	pid_t pid,pid1;

	pid=fork();
	if(pid<0){
		fprintf(stderr,"Fork Failed");
		return 1;
	}
	else if(pid==0){
		pid1=getpid();
		printf("child: pid = %d \n",pid);
		printf("child: pid1 = %d \n",pid1);

	}

	else{
		pid1=getpid();
		printf("parent: pid = %d \n",pid);
		printf("parent: pid1 = %d \n",pid1);



	}
	return 0;


}
```

代码 3

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




int global = 2004;
```

```
int main(){

	pid_t pid,pid1;

	pid=fork();
	if(pid<0){
		fprintf(stderr,"Fork Failed");
		return 1;
	}
	else if(pid==0){
		pid1=getpid();
		printf("child: pid = %d \n",pid);
		printf("child: pid1 = %d \n",pid1);
		global=24;
		printf("child's global = %d , add= %d \n",global,&global);


	}

	else{
		pid1=getpid();
		printf("parent: pid = %d \n",pid);
		printf("parent: pid1 = %d \n",pid1);
		wait(NULL);
		global=88;
		printf("parent's global = %d , add =%d \n",global,&global);


	}
	return 0;


}
```

代码 4:

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




int global = 2004;
int main(){

	pid_t pid,pid1;
```

```
    pid=fork();
    if(pid<0){
        fprintf(stderr,"Fork Failed");
        return 1;
    }
    else if(pid==0){
        pid1=getpid();
        printf("child: pid = %d \n",pid);
        printf("child: pid1 = %d \n",pid1);
        global=24;
        printf("child's global = %d , add = %d \n",global, &global);


    }

    else{
        pid1=getpid();
        printf("parent: pid = %d \n",pid);
        printf("parent: pid1 = %d \n",pid1);
        wait(NULL);
        global=88;
        printf("parent's global = %d , add = %d \n",global, &global);


    }
    global=2004+pid;
    printf("now global (before return) = %d \n",global);
    return 0;


}
```

代码 5：

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

int global = 2004;
int main() {

  pid_t pid, pid1;

  pid = fork();
```

```
 if (pid < 0) {
   fprintf(stderr, "Fork Failed");
   return 1;
 } else if (pid == 0) {
   pid1 = getpid();
   printf("child: pid = %d \n", pid);
   printf("child: pid1 = %d \n", pid1);
   global = 24;
   printf("child's global = %d , add = %d \n", global, &global);

   //system("~/git/oslab-XJTU/1/1.1/system_call");
   system("./system_call");
   return 1;

 }

 else {
   pid1 = getpid();
   printf("parent: pid = %d \n", pid);
   printf("parent: pid1 = %d \n", pid1);
   wait(NULL);
   global = 88;
   printf("parent's global = %d , add = %d \n", global, &global);
 }
 global = 2004;
 printf("now global = %d \n", global);
 return 0;
}
```

代码 6：

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




int global = 2004;
int main(){

      pid_t pid,pid1;

      pid=fork();
      if(pid<0){
            fprintf(stderr,"Fork Failed");
            return 1;
```

```
    }
    else if(pid==0){
        pid1=getpid();
        printf("child: pid = %d \n",pid);
        printf("child: pid1 = %d \n",pid1);
        global=24;
        printf("child's global = %d, add = %d\n",global, &global);
/*
        execlp("/bin/sh","sh","-c","echo Process ID: $$",NULL);
        perror("execlp failed.");
        return 1;
*/


        if(execl("/bin/sudo","sudo","./execl",NULL)==-1){

            perror("Execl failed. \n");
        }
        //if(execl("/bin/sudo","sudo","./system_call",NULL)==-1){

        //      perror("Execl failed. \n");
        //}


    }

    else{
        pid1=getpid();
        printf("parent: pid = %d \n",pid);
        printf("parent: pid1 = %d \n",pid1);
        wait(NULL);
        global=88;
        printf("parent's global = %d ,add = %d \n",global, &global);


    }
    global=2004;
    printf("now global = %d \n",global);
    return 0;


}
```

代码：system_call.c

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


int main(){
	/*
	execlp("/bin/sh","sh","-c","echo Process ID: $$",NULL);
		perror("execlp failed.");
		return 1;
	*/
	pid_t pid=getpid();
	printf("System call Process ID: %d \n",pid);
	return 0;
}
```

代码：execl.c

```
#include <stdio.h>

int main() {

  printf("execl.\n");
  return 0;
}
```

线程相关实验代码

代码 1：

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

int global=2024;


void* threadFunction1(void* arg){
	for(int i=0;i<5001;i++){
		global++;
	}
	printf("global(thread1) = %d \n",global);
	return NULL;
```

```
}

void* threadFunction2(void* arg){
	for(int i=0;i<5001;i++){
		global-=2;
	}
	printf("global(thread2) = %d \n",global);
	return NULL;
}
int main(){

	pthread_t thread1,thread2;
	if(pthread_create(&thread1,NULL,threadFunction1,NULL)!=0){
		perror("Failed to create thread1. \n");
		return 1;
	}

	if(pthread_create(&thread2,NULL,threadFunction2,NULL)!=0){
		perror("Failed to create thread2. \n");
		return 2;
	}

	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);

	printf("finally, global is: %d \n",global);

	return 0;
}
```

代码 2：

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

int global=2024;

pthread_mutex_t lock;
void* threadFunction1(void* arg){
	for(int i=0;i<5001;i++){

		pthread_mutex_lock(&lock);
		global++;
		pthread_mutex_unlock(&lock);
	}
	printf("global(thread1) = %d \n",global);
	return NULL;
```

```
}

void* threadFunction2(void* arg){
	for(int i=0;i<5001;i++){
		pthread_mutex_lock(&lock);
		global-=2;
		pthread_mutex_unlock(&lock);
	}
	printf("global(thread2) = %d \n",global);
	return NULL;
}



int main(){

	pthread_t thread1,thread2;

	pthread_mutex_init(&lock,NULL);

	if(pthread_create(&thread1,NULL,threadFunction1,NULL)!=0){
		perror("Failed to create thread1. \n");
		return 1;
	}

	if(pthread_create(&thread2,NULL,threadFunction2,NULL)!=0){
		perror("Failed to create thread2. \n");
		return 2;
	}

	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);


	pthread_mutex_destroy(&lock);
	printf("finally, global is: %d \n",global);

	return 0;
}
```

代码 3：

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

```
#include<pthread.h>
#include<sys/syscall.h>


void* threadFunction1(void* arg){
	system("./system_call");

	printf("thread1 pid=%d,tid=%d \n",getpid(),syscall(SYS_gettid));
	return NULL;
}
void* threadFunction2(void* arg){
	system("./system_call");

	printf("thread2 pid=%d,tid=%d \n",getpid(),syscall(SYS_gettid));
	return NULL;
}



int main(){
	pthread_t thread1,thread2;

	if(pthread_create(&thread1,NULL,threadFunction1,NULL)!=0){
		perror("Failed to create thread1. \n");
		return 1;
	}
	if(pthread_create(&thread2,NULL,threadFunction2,NULL)!=0){
		perror("Failed to create thread2. \n");
		return 2;
	}

	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);
	return 0;


}
```

代码 4：

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>
#include<pthread.h>
#include<sys/syscall.h>
```

```
void* threadFunction1(void* arg){
	printf("thread1 pid=%d,tid=%d \n",getpid(),syscall(SYS_gettid));

	execl("/bin/sudo","sudo","./system_call",NULL);
	//execl("sudo ./root/1.2/system_call","system_call",NULL);
		return NULL;
}
void* threadFunction2(void* arg){
	printf("thread2 pid=%d,tid=%d \n",getpid(),syscall(SYS_gettid));

	execl("/bin/sudo","sudo","./system_call",NULL);
	//execl("sudo ./root.1.2/system_call","system_call",NULL);
	return NULL;
}



int main(){
	pthread_t thread1,thread2;

	if(pthread_create(&thread1,NULL,threadFunction1,NULL)!=0){
		perror("Failed to create thread1. \n");
		return 1;
	}
	if(pthread_create(&thread2,NULL,threadFunction2,NULL)!=0){
		perror("Failed to create thread2. \n");
		return 2;
	}

	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);
	return 0;


}
```

代码 system_call.c

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<pthread.h>
#include<sys/syscall.h>
int main(){
	/*
	execlp("/bin/sh","sh","-c","echo Process ID: $$",NULL);
		perror("execlp failed.");
		return 1;
```

```
    */
    pid_t pid=getpid();
    printf("System call Process ID: %d \n",pid);
    //pthread_t tid=syscall(SYS_gettid);
    //printf("System call Thread ID: %d \n",tid);
    return 0;
}
```

自旋锁相关实验

代码：

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

typedef struct{
    volatile 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)==1){


    }

}


void spinlock_unlock(spinlock_t *lock){

    __sync_lock_release(&lock->flag);
}

int shared_value=0;

void* thread_function1(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;

}
void* thread_function2(void* arg){

    spinlock_t *lock=(spinlock_t *)arg;
    for(int i=0;i<5000;++i){
        spinlock_lock(lock);
        shared_value-=2;
        spinlock_unlock(lock);
    }

    return NULL;

}
int main(){
    pthread_t thread1,thread2;
    spinlock_t lock;

    printf("shared value = %d \n",shared_value);

    spinlock_init(&lock);
    //spinlock_lock(&lock);

    if(pthread_create(&thread1,NULL,thread_function1,&lock)==-1){
        printf("create thread1 failed.");
        return 1;
    }
    if(pthread_create(&thread2,NULL,thread_function2,&lock)==-1){
        printf("create thread2 failed.");
        return 2;
    }

    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);


    //spinlock_unlock(&lock);

    printf("shared value = %d \n",shared_value);

    return 0;
}
```

1.8.2 附件 2 Readme

readme 附于文件最后。

2 进程通信与内存管理

2.1.1 进程的软中断通信 实验目的

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 实验内容

（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.

2.1.3 实验思想

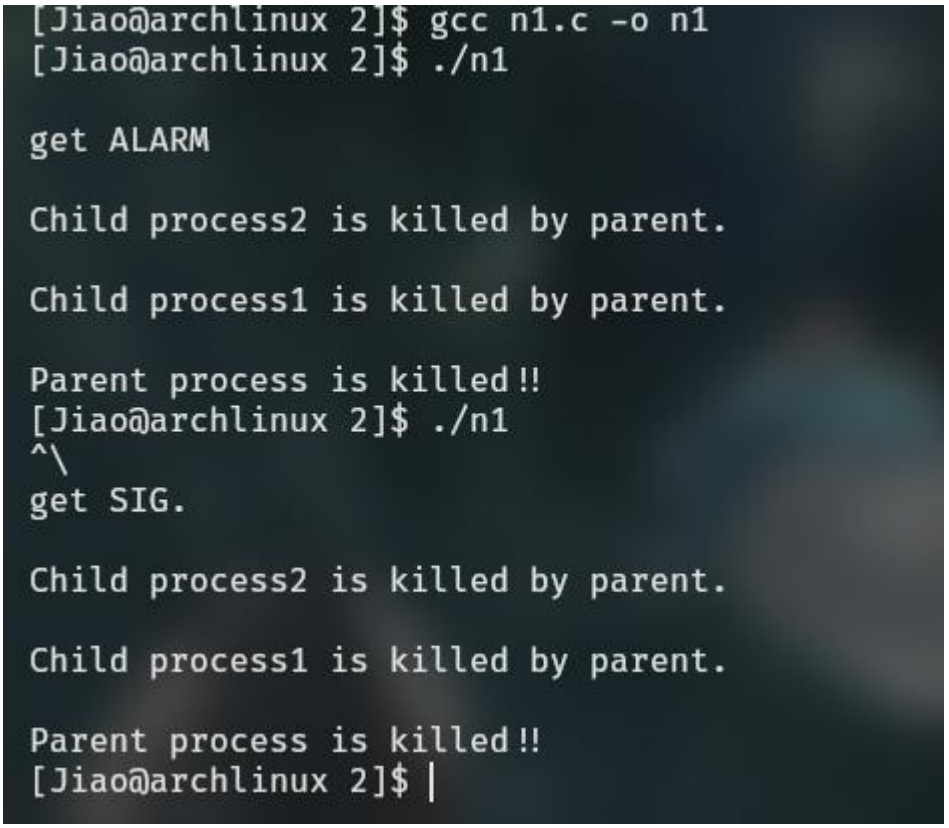
通过合理使用 kill、signal 函数，实现进程间的控制与通信，实现特定进程收到特定信号后做出特定行为。

2.1.4 测试数据设计

不需要设定测试数据。

2.1.5 程序运行初值及运行结果分析

以下为运行结果：



运行结果分析：

第一次运行：由于 5s 之内没有进行任何操作，所以父进程接收到 ALARM 信号，相关 handler 操作被执行，导致两子进程被终止。

第二次运行：由于在 5s 之内通过键盘输入了 SIGQUIT 信号，被父进程接收，相关的 hander 操作被执行，导致两个子进程被提前终止。

2.2.1 进程管道通信 实验目的

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。

2.2.2 实验内容

（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 2000 个字符，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 实验思想

通过 pipe 实现管道通信，两个子进程共同向管道写入数据。观察不同互斥锁情况下管道中数据的情况。

2.2.4 测试数据设计

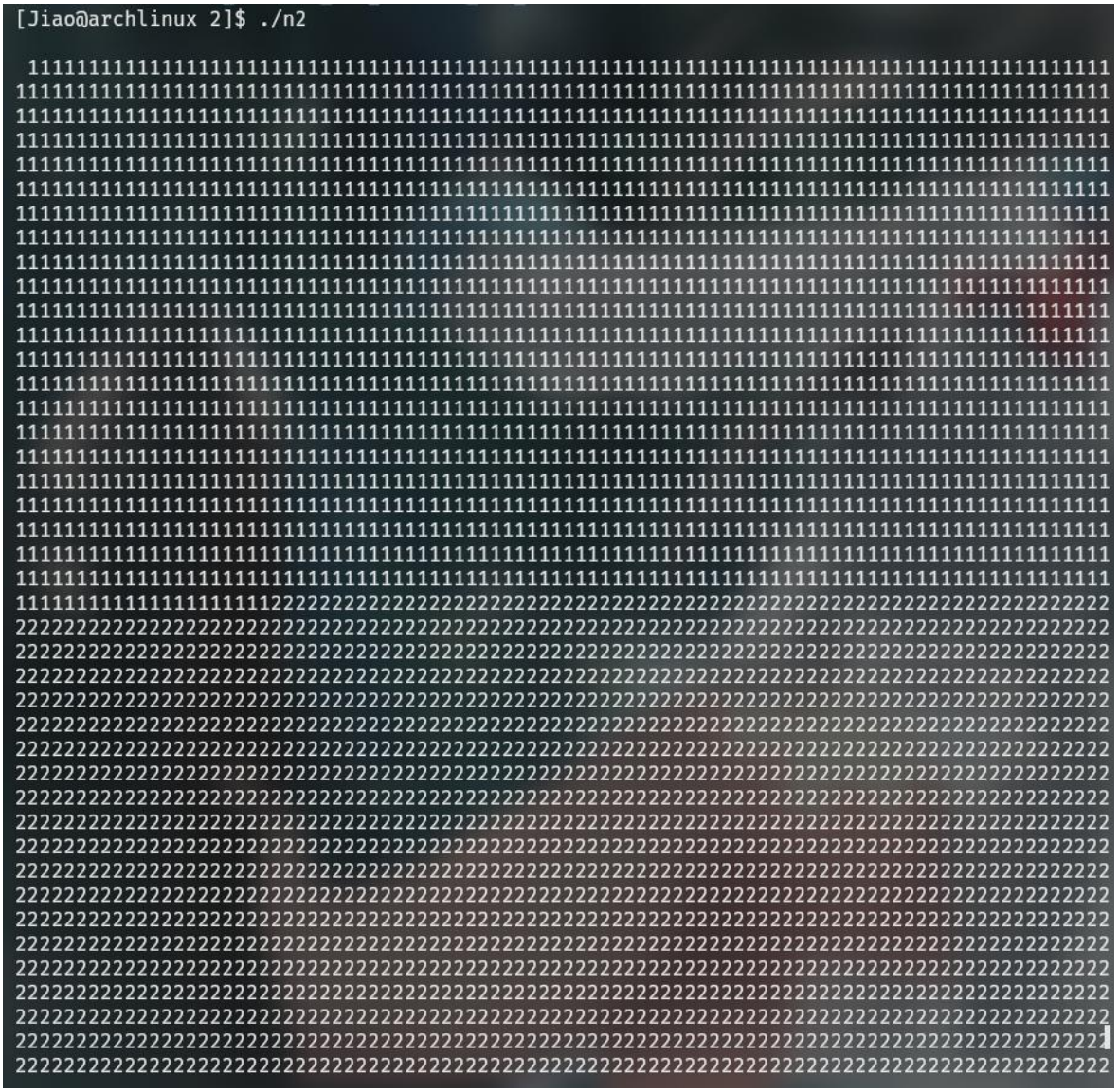
本实验无需设计测试数据。

2.2.5 程序运行初值及运行结果分析

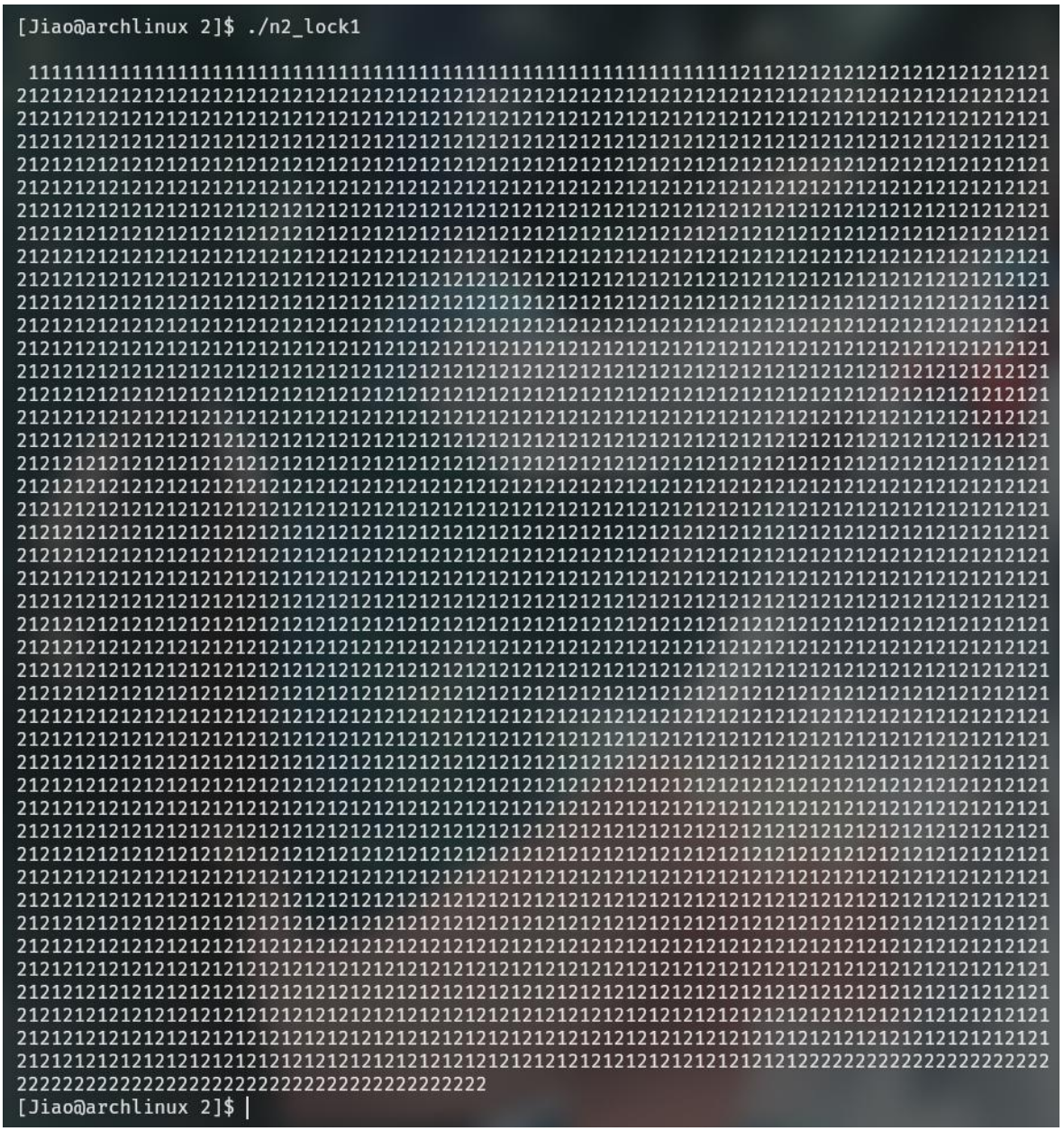
在一个进程中创建两个子进程，分别让两个子进程在管道中写入 1/2，在子进程运行结束后，让父进程在管道中读出数据并输出，观察结果。

运行结果如下：

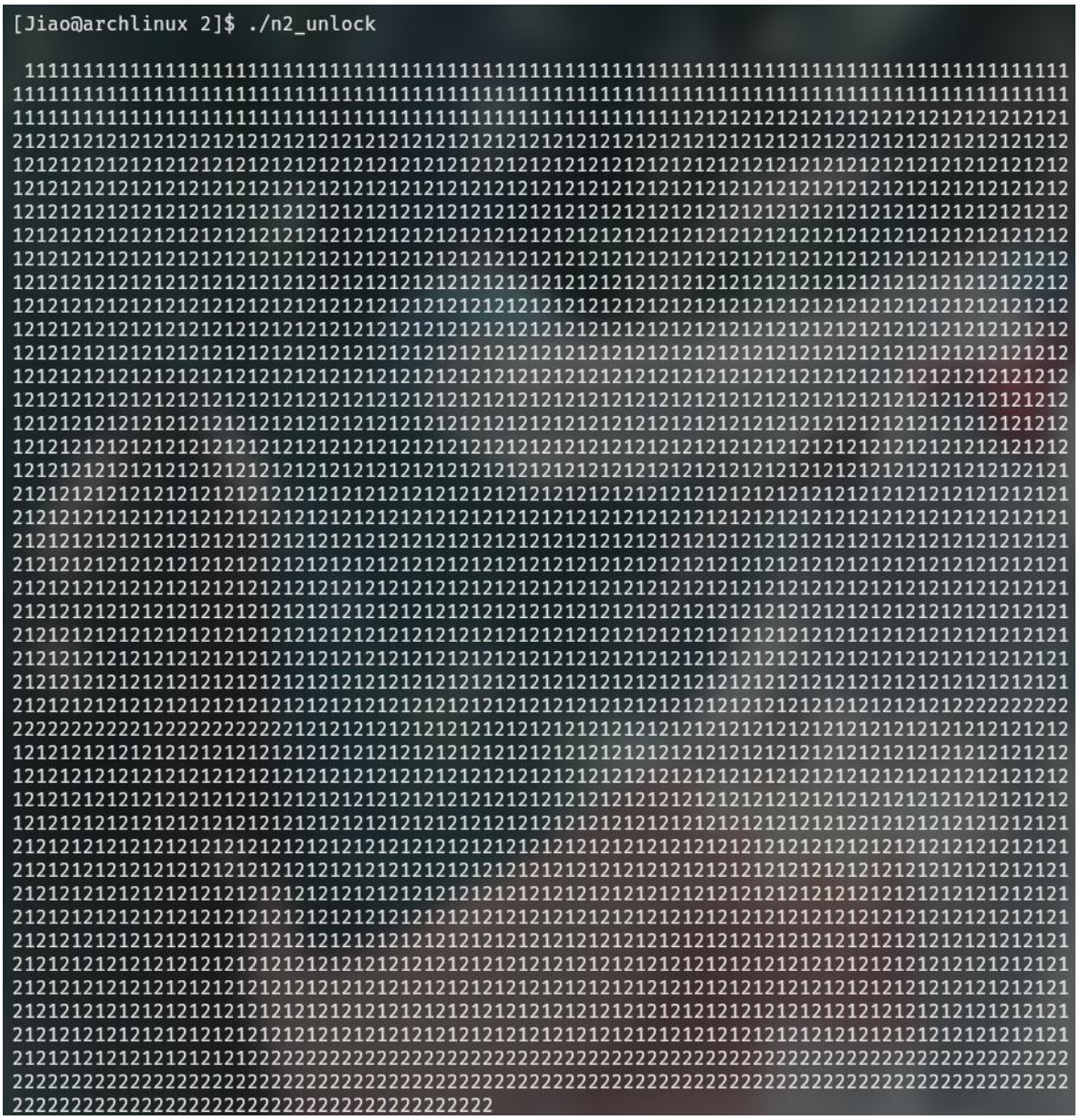
加入两个互斥锁：



加入一个互斥锁：



不加互斥锁：



运行结果分析：

加入两个互斥锁后，可以做到让一个子进程写入完成后，第二个子进程才开始写入，这样就可以保证写入个数据是有序的。

如果没有加两个互斥锁，两个进程会 concurrently 地运行，就会导致写入的数据是混乱无序的。

2.3.1 页面置换 实验目的

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。

2.3.2 实验内容

（1）理解页面置换算法 FIFO、LRU 的思想及实现的思路。

（2）参考给出的代码思路，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。

2.3.3 实验思想

通过模拟页面置换，深入了解页面置换的相关算法，比较页面置换算法之间的差异。

2.3.4 测试数据设计

需要设计一组 belady 现象的输入数据，

如下：

```
9       bool flag;
10      unsigned int counter;
11    } page;
12
13    page *page_table = NULL;
14    unsigned int *reference = NULL;
15    unsigned int table_size = 10;
16    unsigned int frame_size = 5;
17    unsigned int ref_size = 20;
18    unsigned int belady[12] = {1, 2, 3, 4, 1, 2, 5, 5, 5, 5, 5, 5};
19    unsigned int belady2[13] = {1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3};
20
```

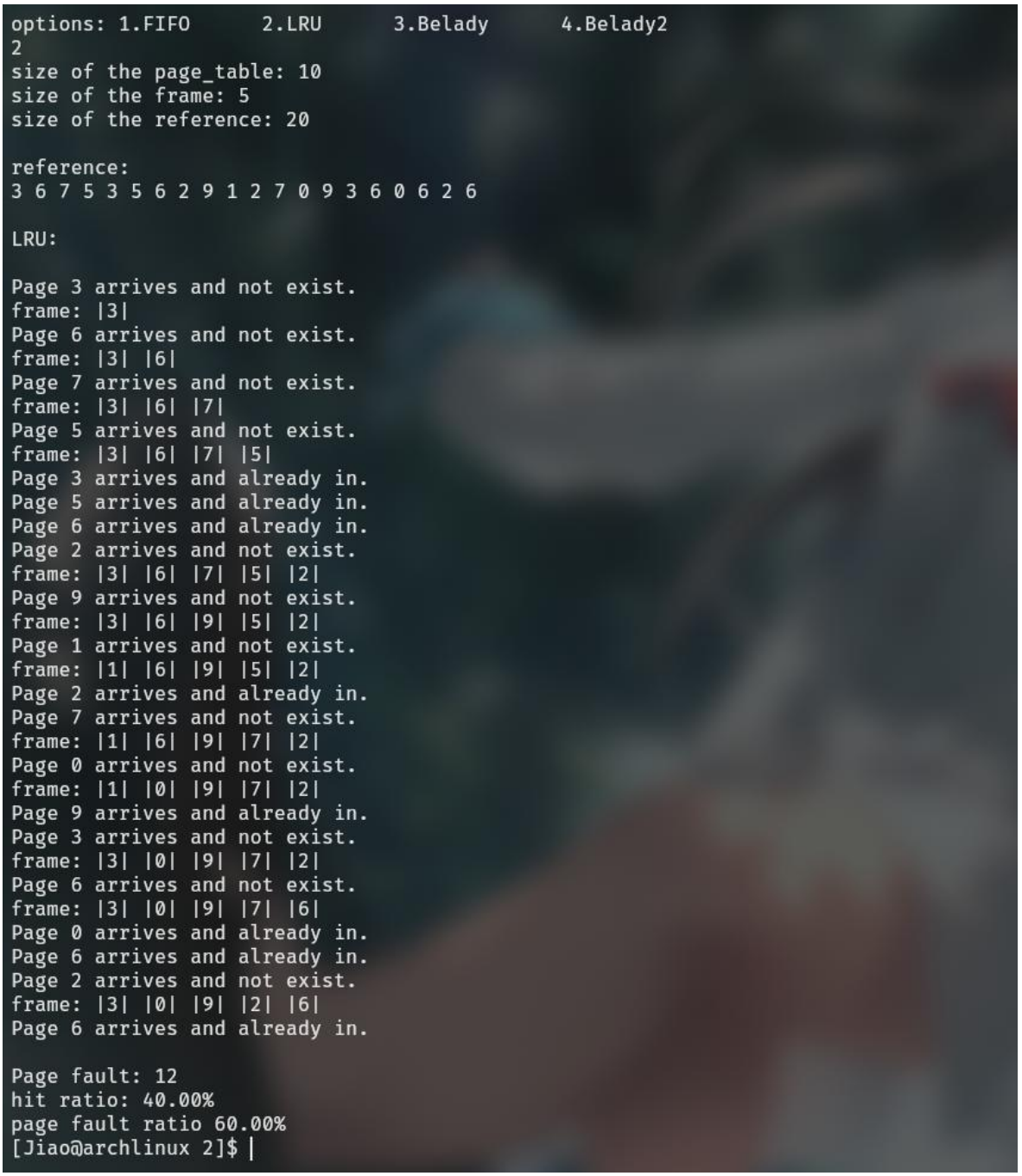
2.3.5 程序运行初值及运行结果分析

运行结果如下：

FIFO：

```
options: 1.FIFO      2.LRU      3.Belady      4.Belady2
1
size of the page_table: 10
size of the frame: 5
size of the reference: 20

reference:
3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6

FIFO:

Page 3 arrives and not exist.
frame: |3|
Page 6 arrives and not exist.
frame: |3| |6|
Page 7 arrives and not exist.
frame: |3| |6| |7|
Page 5 arrives and not exist.
frame: |3| |6| |7| |5|
Page 3 arrives and already in.
Page 5 arrives and already in.
Page 6 arrives and already in.
Page 2 arrives and not exist.
frame: |3| |6| |7| |5| |2|
Page 9 arrives and not exist.
frame: |6| |7| |5| |2| |9|
Page 1 arrives and not exist.
frame: |7| |5| |2| |9| |1|
Page 2 arrives and already in.
Page 7 arrives and already in.
Page 0 arrives and not exist.
frame: |5| |2| |9| |1| |0|
Page 9 arrives and already in.
Page 3 arrives and not exist.
frame: |2| |9| |1| |0| |3|
Page 6 arrives and not exist.
frame: |9| |1| |0| |3| |6|
Page 0 arrives and already in.
Page 6 arrives and already in.
Page 2 arrives and not exist.
frame: |1| |0| |3| |6| |2|
Page 6 arrives and already in.

Page fault: 11
hit ratio: 45.00%
page fault ratio 55.00%
[Jiao@archlinux 2]$
```

LRU：


options: 1.FIFO 2.LRU 3.Belady 4.Belady2
2
size of the page_table: 10
size of the frame: 5
size of the reference: 20
reference:
3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6
LRU:
Page 3 arrives and not exist.
frame: |3|
Page 6 arrives and not exist.
frame: |3| |6|
Page 7 arrives and not exist.
frame: |3| |6| |7|
Page 5 arrives and not exist.
frame: |3| |6| |7| |5|
Page 3 arrives and already in.
Page 5 arrives and already in.
Page 6 arrives and already in.
Page 2 arrives and not exist.
frame: |3| |6| |7| |5| |2|
Page 9 arrives and not exist.
frame: |3| |6| |9| |5| |2|
Page 1 arrives and not exist.
frame: |1| |6| |9| |5| |2|
Page 2 arrives and already in.
Page 7 arrives and not exist.
frame: |1| |6| |9| |7| |2|
Page 0 arrives and not exist.
frame: |1| |0| |9| |7| |2|
Page 9 arrives and already in.
Page 3 arrives and not exist.
frame: |3| |0| |9| |7| |2|
Page 6 arrives and not exist.
frame: |3| |0| |9| |7| |6|
Page 0 arrives and already in.
Page 6 arrives and already in.
Page 2 arrives and not exist.
frame: |3| |0| |9| |2| |6|
Page 6 arrives and already in.
Page fault: 12
hit ratio: 40.00%
page fault ratio 60.00%
[Jiao@archlinux 2]$

Belady 现象展示：

```
[Jiao@archlinux 2]$ ./3
page replacement

options: 1.FIFO      2.LRU      3.Belady      4.Belady2
3

reference:
1 2 3 4 1 2 5 5 5 5 5 5

FIFO(Belady):

Page 1 arrives and not exist.
frame: |1|
Page 2 arrives and not exist.
frame: |1| |2|
Page 3 arrives and not exist.
frame: |1| |2| |3|
Page 4 arrives and not exist.
frame: |2| |3| |4|
Page 1 arrives and not exist.
frame: |3| |4| |1|
Page 2 arrives and not exist.
frame: |4| |1| |2|
Page 5 arrives and not exist.
frame: |1| |2| |5|
Page 5 arrives and already in.
Page 5 arrives and already in.
Page 5 arrives and already in.
Page 5 arrives and already in.
Page 5 arrives and already in.

Page fault: 7
hit ratio: 41.67%
page fault ratio 58.33%
[Jiao@archlinux 2]$
```

```
[Jiao@archlinux 2]$ ./3
page replacement

options: 1.FIFO      2.LRU      3.Belady      4.Belady2
4

reference:
1 2 3 4 5 1 2 3 4 5 1 2 3

reference:
1 2 3 4 5 1 2 3 4 5 1 2 3

FIFO(Belady2):

Page 1 arrives and not exist.
frame: |1|
Page 2 arrives and not exist.
frame: |1| |2|
Page 3 arrives and not exist.
frame: |1| |2| |3|
Page 4 arrives and not exist.
frame: |1| |2| |3| |4|
Page 5 arrives and not exist.
frame: |2| |3| |4| |5|
Page 1 arrives and not exist.
frame: |3| |4| |5| |1|
Page 2 arrives and not exist.
frame: |4| |5| |1| |2|
Page 3 arrives and not exist.
frame: |5| |1| |2| |3|
Page 4 arrives and not exist.
frame: |1| |2| |3| |4|
Page 5 arrives and not exist.
frame: |2| |3| |4| |5|
Page 1 arrives and not exist.
frame: |3| |4| |5| |1|
Page 2 arrives and not exist.
frame: |4| |5| |1| |2|
Page 3 arrives and not exist.
frame: |5| |1| |2| |3|

Page fault: 13
hit ratio: 0.00%
page fault ratio 100.00%
[Jiao@archlinux 2]$
```

运行结果分析：

通过随机生成需要的 reference，模拟 FIFO 和 LRU 算法的执行过程，并计算缺页率，可以发现。相同的条件下，一般来讲 LRu 算法优于 FIFO 算法，可以有更小的缺页率。

通过用 FIFO 算法运行设计好的两组 reference，我们可以发现，Belady2 的 frame 数量大于 Belady1，但是缺页率反而会比 Belady1 要高，这就是 FIFO 算法可能出现的 Belady 现象。

2.7 页面置换算法复杂度分析

FIFO 只需要维护一个队列，其复杂度等于队列复杂度。

LRU 相当于查找替换，是 n^2 的复杂度。

2.8 回答问题

软中断通信

（1）.你最初认为运行结果会怎么样？写出你猜测的结果。

父进程会在子进程结束之后才结束，无论是通过接收 ALARM 还是 SIGQUIT，都是在接收相关信号之后，令两个子进程终止之后才会结束运行。

（2）实际的结果什么样？有什么特点？在接收不同中断前后有什么差别？请将 5 秒内中断 和 5 秒后中断的运行结果截图，试对产生该现象的原因进行分析。

运行结果截图已经在上面给出过。

实际运行结果正如猜测一致。

接收不同信号会导致不同的 handler 被触发。

（3）改为闹钟中断后，程序运行的结果是什么样子？与之前有什么不同？

为了区分两种信号，我在两个不同的 handler 中附加了 printf 语句，输出以显示到底是哪个 handler 被触发。

除了显示的触发 handler 不同以外，结果基本一致。

（4）kill 命令在程序中使用了几次？每次的作用是什么？执行后的现象是什么？

四次。

两次是由 handler 发出，用于终止两个子进程；

两次是由子进程发出，用于通知父进程子进程已经准备好接收信号了。

由 handler 发出的信号，执行后，两子进程终止；由子进程发出的信号执行后，会输出子进程准备好接收信号了（现在已经被注释了，因为其实没有必要进行输出。）

（5）使用 kill 命令可以在进程的外部杀死进程。进程怎样能主动退出？这两种退出方式哪种更好一些？

进程除了通过 kill 被从外部杀死，也可以在内部通过 return 返回或者 exit 强制退出来终止。

return 和 exit 属于进程主动退出，一般是进程完成任务后退出或者检测到错误后退出，可以保证进程完成自己的工作后才会被结束，能够保证进程工作的完整性和可靠性。

kill 等属于通过过外部信号退出，这可以实现一个进程控制其他进程退出，相对来讲更加的灵活，能够适应复杂的编程需求。

（7）父进程向子进程发送信号时，如何确保子进程已经准备好接收信号？

这个问题在（4）中已经回答过了，子进程在准备好接收信号之后才向父进程发送信号通知父进程，即可保证父进程向子进程发送信号时，子进程已经准备好接收信号。

（8）如何阻塞住子进程，让子进程等待父进程发来信号？

可以定义判断条件，在接收到父进程信号之前，子进程保持循环且不做任何操作。

管道通信

（1）你最初认为运行结果会怎么样？
加锁的程序会严格的先 1 后 2 写入，不加（两个）锁的程序会以无序的顺序写入数据。

（2）实际的结果什么样？有什么特点？试对产生该现象的原因进行分析。
对于加入两个互斥锁的程序，首先申请到互斥锁的进程会优先完成他的写入，在他写入完成之后，才会释放互斥锁，这时才会让第二个进程进行写入，因此从管道中读取的信息将会是有序的写入。
对于只加一个或者不加互斥锁的程序，当一个进程正在写入时，另一个不加锁的进程也会在管道中进行写操作，这就会导致输出结果无序。

（3）实验中管道通信是怎样实现同步与互斥的？如果不控制同步与互斥会发生什么后果？
通过加互斥锁的方式控制写入的同步和互斥。

如果不保证写入的同步和互斥的话就会得到混乱的数据。

页面置换

（1）比较 FIFO 算法和 LRU 算法。

FIFO 算法实现比 LRU 算法简单，相当于只需要维持一个队列即可；

LRU 算法则需要为每个页面多记录一个最近一次使用的时间，这确实会增大系统开销，

但是，LRU 由于符合程序运行的时间局部性和空间局部性，可以获得较高的页面命中率，缺页率较低，在这个意义上加快了程序运行，减小了多次访问磁盘的开销，拥有更优秀的性能。

（2）LRU 算法是基于程序的局部性原理而提出的算法，你模拟实现的 LRU 算法有没有体现

出该特点？如果有，是如何实现的？

我设计的 LRU 算法会在参考队列中将随机生成的 reference 进行修改，实现类似于局部性的 reference 的生成。

（3）在设计内存管理程序时,应如何提高内存利用率。

要充分利用程序的局部性原理，综合考虑，寻找最优算法。

2.9 实验总结

2.9.1 实验中的问题与解决过程

问题：

在管道通信中，无法控制通信的同步与互斥。

解决方式：

当子进程对管道进行写操作时，加互斥锁并且关闭父进程对管道的写通道。

2.9.2 实验收获

通过本次实验，我对进程间通信的原理和实现方式有了更深的理解，掌握了进程软中断通信和管道通信的实现方式；通过模拟 FIFO 和 LRU 页面置换算法，对于操作系统管理页面的实现原理有了深刻的理解，并且对这两种算法的优劣进行了比对，理解了在不同的情况下使用何种算法。

2.9.3 意见与建议

希望可以完善实验指导文档，指导文档中有部分表述模糊希望可以替换成更加清晰的表述。

2.10 附件

代码 1，软中断通信

```
#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>
```

```
#include <signal.h>

#include<time.h>

int flag = 1;

pid_t pid1 = -1, pid2 = -1;

void parent() {

   if(pid1>0 && pid2>0)

   {

   printf("\nget SIG.\n");

   kill(pid1,16);

   kill(pid2,17);

   }

}

void alrm_handler() {

   printf("\nget ALARM\n");

      kill(pid1, 16);

      kill(pid2, 17);

}

void child1(){

   printf("\nChild process1 is killed by parent.\n");

   flag=0;

}

void child2(){

   printf("\nChild process2 is killed by parent.\n");

   flag=0;

}
```

```
void child_handler(int signum) {
    if (signum == SIGUSR1) {
        //printf("\nChild1 process is ready to receive signals.\n");
    }
    else if(signum == SIGUSR2){
        //printf("\nChild2 process is ready to receive signals.\n");
    }
}

int main() {
    signal(SIGUSR1, child_handler);
    signal(SIGUSR2, child_handler);
    signal(SIGINT, parent);
    signal(SIGQUIT, parent);
    while (pid1 == -1){
        pid1 = fork();
    }

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

        if (pid2 > 0){
          //pause();
```

```
        //pause();
        //alarm(5);
        signal(SIGALRM, alrm_handler);
        alarm(5);
        wait(NULL);
        wait(NULL);
        printf("\nParent process is killed!!\n");
      }
      else {
        //sleep(1);
        signal(17,child2);
        kill(getppid(),SIGUSR2);
        while(flag)pause();
        return 0;
      }
    }
    else{
      //sleep(2);
      signal(16,child1);
      kill(getppid(),SIGUSR1);
      while(flag)pause();
      return 0;
    }
    return 0;
}
```

代码 2，管道通信：

```
#include<unistd.h>
#include<signal.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<time.h>
#include<stdio.h>
int pid1 = -1,pid2 = -1;



int main(){

  int fd[2];
  char InPipe[4000];
  char c1 = '1',c2 = '2';
  pipe(fd);

  while(pid1 == -1){
    pid1 = fork();
  }
  if(pid1 == 0){
    close(fd[0]);
    lockf(fd[1],1,0);

    for(int i=0;i<2000;i++){
```

```
    write(fd[1],&c1,1);
  }

  sleep(5);
  lockf(fd[1],0,0);
  close(fd[1]);
}
else{
  while(pid2 == -1){
    pid2 = fork();
  }
  if(pid2 == 0){
    close(fd[0]);
    lockf(fd[1],1,0);
    for(int i=0;i<2000;i++){
      write(fd[1],&c2,1);
    }
    sleep(5);
    lockf(fd[1],0,0);
    close(fd[1]);
  }
  else{
    wait(NULL);
    wait(NULL);
```

```
    close(fd[1]);


    int read_content = read(fd[0],InPipe,4000);
    InPipe[read_content] = '\0';
    printf("\n %s \n", InPipe);
    close(fd[0]);
    exit(0);
   }
  }


  exit(0);
}
```

代码 3，页面置换：

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct page {
  unsigned int frame;
  bool flag;
  unsigned int counter;
```

```
} page;


page *page_table = NULL;
unsigned int *reference = NULL;
unsigned int table_size = 10;
unsigned int frame_size = 5;
unsigned int ref_size = 20;
unsigned int belady[12] = {1, 2, 3, 4, 1, 2, 5, 5, 5, 5, 5, 5};
unsigned int belady2[13] = {1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3};


void initial_page_table() {
  page_table = (page *)malloc(table_size * sizeof(page));
  if (!page_table) {
    printf("page_table: malloc failed!\n");
    exit(0);
  }
  for (unsigned int i = 0; i < table_size; i++) {
    page_table[i].flag = false;
    page_table[i].counter = 0;
  }
}


void initial_reference() {
  reference = (unsigned int *)malloc(ref_size * sizeof(unsigned int));
  if (!reference) {
```

```
    printf("reference: malloc failed!\n");
    exit(0);
  }
  // srand((unsigned int)time(0));
  for (unsigned int i = 0; i < ref_size; i++)
    reference[i] = rand() % table_size;
}

void print_reference() {
  printf("\nreference:\n");
  for (unsigned int i = 0; i < ref_size; i++) {
    printf("%u ", reference[i]);
  }
  printf("\n");
}

void release() {
  if (page_table)
    free(page_table);
  if (reference && reference != belady)
    free(reference);
}

void Belady(void) {
  ref_size = 12;
```

```
  frame_size = 3;
  table_size = 6;
  reference = belady;
}
void Belady2(void) {
  ref_size = 13;
  frame_size = 4;
  table_size = 6;
  reference = belady2;
}

int FIFO(void) {

  unsigned int diseffect = 0;
  unsigned int front = 0;
  unsigned int rear = 0;
  unsigned int queue_size = frame_size + 1;
  unsigned int *queue =
      (unsigned int *)malloc(queue_size * sizeof(unsigned int));
  if (!queue) {
    printf("queue: malloc failed!\n");
    exit(0);
  }

  for (unsigned int i = 0; i < ref_size; i++) {
```

```
  printf("Page %u arrives ", reference[i]);


  if (!page_table[reference[i]].flag) {
    printf("and not exist.\n");
    diseffect++;
    if ((rear + 1) % queue_size == front) {


      page_table[queue[front]].flag = false;
      front = (front + 1) % queue_size;
    }


    page_table[reference[i]].flag = true;
    page_table[reference[i]].frame = rear;
    queue[rear] = reference[i];
    rear = (rear + 1) % queue_size;


    printf("frame: ");
    for (unsigned int k = front; k != rear; k = (k + 1) % queue_size)
      printf("|%u| ", queue[k]);
    printf("\n");
  } else {


    printf("and already in.\n");
  }
}
```

```
  free(queue);
  return diseffect;
}

int find_min(unsigned int *frame) {
  unsigned int min = UINT_MAX, frame_num = 0;
  for (unsigned int i = 0; i < frame_size; i++) {
    if (page_table[frame[i]].counter < min) {
      min = page_table[frame[i]].counter;
      frame_num = i;
    }
  }
  return frame_num;
}

unsigned int LRU(void) {
  unsigned int diseffect = 0;
  unsigned int clock = 0;
  unsigned int full = 0;
  unsigned int *frame =
      (unsigned int *)malloc(frame_size * sizeof(unsigned int));
  if (!frame) {
    printf("frame: malloc failed!\n");
    exit(0);
```

```
}


for (unsigned int i = 0; i < ref_size; i++) {
  printf("Page %u arrives ", reference[i]);


  if (clock == UINT_MAX) {
    unsigned int min;
    unsigned int *temp =
        (unsigned int *)malloc(frame_size * sizeof(unsigned int));
    for (unsigned int i = 0; i < frame_size; i++) {
      min = find_min(frame);
      temp[i] = frame[min];
      page_table[frame[min]].counter = UINT_MAX;
    }
    for (unsigned int i = 0; i < frame_size; i++)
      page_table[temp[i]].counter = i;
    free(temp);
    clock = frame_size;
  }
  page_table[reference[i]].counter = clock;
  clock++;


  if (!page_table[reference[i]].flag) {
    printf("and not exist.\n");
    diseffect++;
```

```
    if (full < frame_size) {

      page_table[reference[i]].flag = true;
      page_table[reference[i]].frame = full;
      frame[full] = reference[i];
      full++;
    } else {
      unsigned int replace = find_min(frame);

      page_table[frame[replace]].flag = false;

      page_table[reference[i]].flag = true;
      page_table[reference[i]].frame = replace;
      frame[replace] = reference[i];
    }

    printf("frame: ");
    for (unsigned int k = 0; k < full; k++)
      printf("|%u| ", frame[k]);
    printf("\n");
  } else {

    printf("and already in.\n");
  }
}
```

```
 free(frame);
 return diseffect;
}

char set_parameter(void) {

 printf("\noptions: 1.FIFO    2.LRU    3.Belady    4.Belady2\n");
 char choice = getchar();
 while (getchar() != '\n')
  ;
 if (choice == '3') {
  Belady();
  return choice;
 }
 if (choice == '4') {
  Belady2();
  // printf("!\n");
  print_reference();
  return choice;
 }

 unsigned int size = 0;

 printf("size of the page_table: ");
```

```
if (scanf("%u", &size) == 1 && size > 0 && size <= 128)
  table_size = size;
else {
  table_size = 10;
  printf("default = 10\n");
}
while (getchar() != '\n')
  ;

printf("size of the frame: ");
if (scanf("%u", &size) == 1 && size > 0 && size <= 32)
  frame_size = size;
else {
  frame_size = 4;
  printf("default = 4\n");
}
while (getchar() != '\n')
  ;

printf("size of the reference: ");
if (scanf("%u", &size) == 1 && size >= 0 && size <= 100)
  ref_size = size;
else {
  ref_size = 20;
  printf("default = 20\n");
```

```
 }
 while (getchar() != '\n')
  ;

 return choice;
}

int main(void) {

 printf("page replacement\n");

 char choice1 = set_parameter();
 if (choice1 != '3' && choice1 != '4') {
  initial_reference();
  // printf("fun");
 }

 unsigned int diseffect;

 while (true) {
  print_reference();
  initial_page_table();

  switch (choice1) {
  case '1':
```

```
  printf("\nFIFO:\n\n");
  diseffect = FIFO();
  //  printf("fifo");

  break;
case '2':
  printf("\nLRU:\n\n");
  diseffect = LRU();
  // printf("lru");
  break;
case '3':
  printf("\nFIFO(Belady):\n\n");
  diseffect = FIFO();
  break;
case '4':
  printf("\nFIFO(Belady2):\n\n");
  diseffect = FIFO();
  break;
default:
  printf("\ndefault = FIFO:\n\n");
  diseffect = FIFO();
}

printf("\nPage fault: %d\n", diseffect);
printf("hit ratio: %.2f%%\n", (1.0 - (float)diseffect / ref_size) * 100.0);
```

```
    printf("page fault ratio %.2f%%\n", ((float)diseffect / ref_size) * 100);
    break;
  }

  return 0;
}
```

2.10.2 附件 2 Readme

readme 附于文件最后。

3 文件系统

3.1 实验目的

通过一个简单文件系统的设计，加深理解文件系统的内部实现原理。

3.2 实验内容

模拟 EXT2 文件系统原理设计实现一个类 EXT2 文件系统。

3.3 实验思想

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。

3.4 实验步骤

定义类 EXT2 文件系统所需的数据结构，包括组描述符、索引结点和目录项。

实现底层函数，包括分配数据块等 操作。

实现命令层函数，包括 dir 等操作。

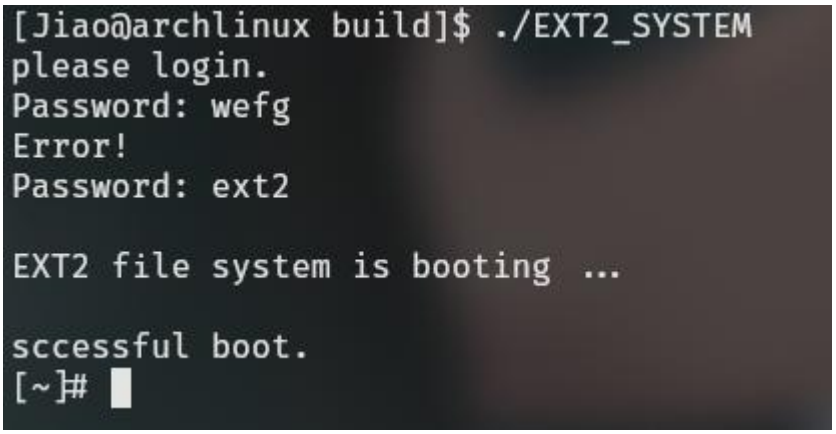
完成 shell 的设计。

测试整个文件系统的功能。

3.5 程序运行初值及运行结果分析

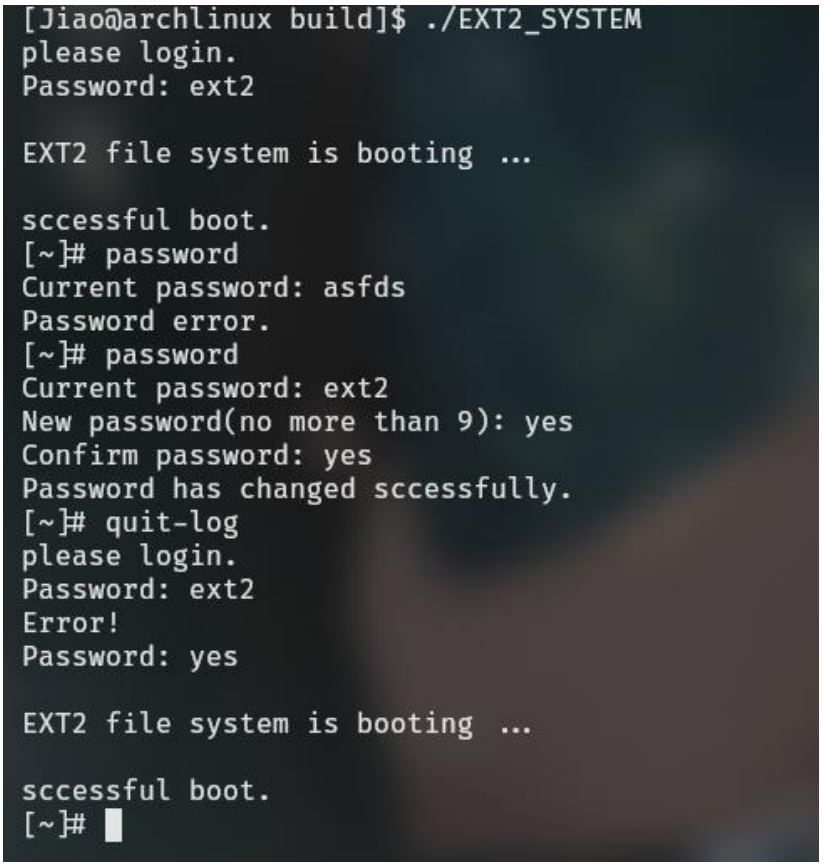
运行结果展示：

（1） Login



输入正确的密码才可以登录成功。

（2） quit-log(relogin)



退出当前用户，也就是回到登录界面。

(3) ls

```
[~]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    17          Wed Dec  4 16:16:12 2024  Wed Dec  4 16:20:33 2024  Wed Dec  4 16:16:12 2024
..        <DIR>   r_w__    17          Wed Dec  4 16:16:12 2024  Wed Dec  4 16:20:33 2024  Wed Dec  4 16:16:12 2024
[~]#
```

列出当前文件夹下文件信息。

(4)mkdir

```
[~]# format
format acomplished.
Volume Name: EXT2FS
Block Size: 512Bytes
Free Block: 4095
Free Inode: 4095
Directories: 1
[~]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    17          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    17          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:55 2024
[~]# mkdir sub1
[~]# mkdir sub2
[~]# mkdir sub3
[~]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    50          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:56 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    50          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:56:56 2024  Thu Dec  5 14:56:55 2024
sub1      <DIR>   r_w__    17          Thu Dec  5 14:56:59 2024  Thu Dec  5 14:56:59 2024  Thu Dec  5 14:56:59 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

创建文件夹。

(5)create(touch)

```
[~]# cd sub1
[~/sub1]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    17          Thu Dec  5 14:56:59 2024  Thu Dec  5 14:56:59 2024  Thu Dec  5 14:56:59 2024
..        <DIR>   r_w__    50          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:57:07 2024  Thu Dec  5 14:56:55 2024
[~/sub1]# create c.exe
[~/sub1]# create c
[~/sub1]# create c
A file with the same name exists.
[~/sub1]# mkdir ss
[~/sub1]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    46          Thu Dec  5 14:56:59 2024  Thu Dec  5 14:57:58 2024  Thu Dec  5 14:56:59 2024
..        <DIR>   r_w__    50          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:57:07 2024  Thu Dec  5 14:56:55 2024
c.exe     <FILE>  r_w_x    0          Thu Dec  5 14:58:05 2024  Thu Dec  5 14:58:05 2024  Thu Dec  5 14:58:05 2024
c         <FILE>  r_w_x    0          Thu Dec  5 14:58:08 2024  Thu Dec  5 14:58:08 2024  Thu Dec  5 14:58:08 2024
ss        <DIR>   r_w__    17          Thu Dec  5 14:58:14 2024  Thu Dec  5 14:58:14 2024  Thu Dec  5 14:58:14 2024
[~/sub1]#
```

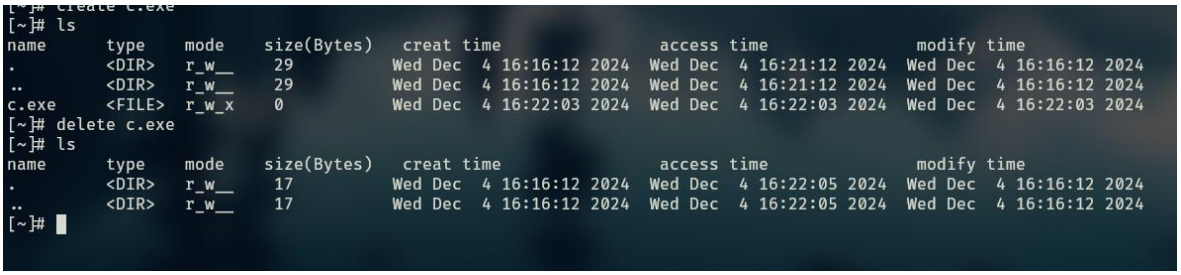
创建文件。

(6)rmdir(delete folder/directory)

```
[~/sub1]# cd ..
[~]# rmdir sub1
[~]# ls
name      type    mode    size(Bytes)   creat time               access time              modify time
.         <DIR>   r_w__    39          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:57:07 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    39          Thu Dec  5 14:56:55 2024  Thu Dec  5 14:57:07 2024  Thu Dec  5 14:56:55 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

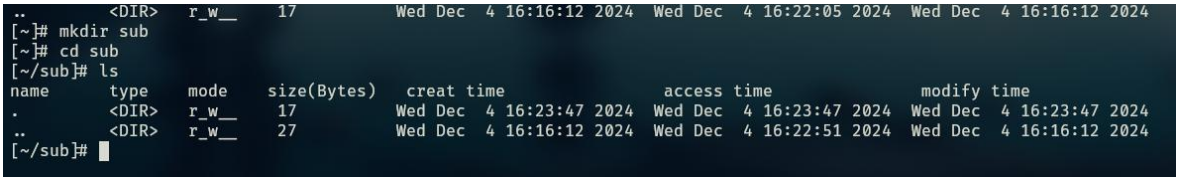
删除文件夹。

(7)delete(rm)



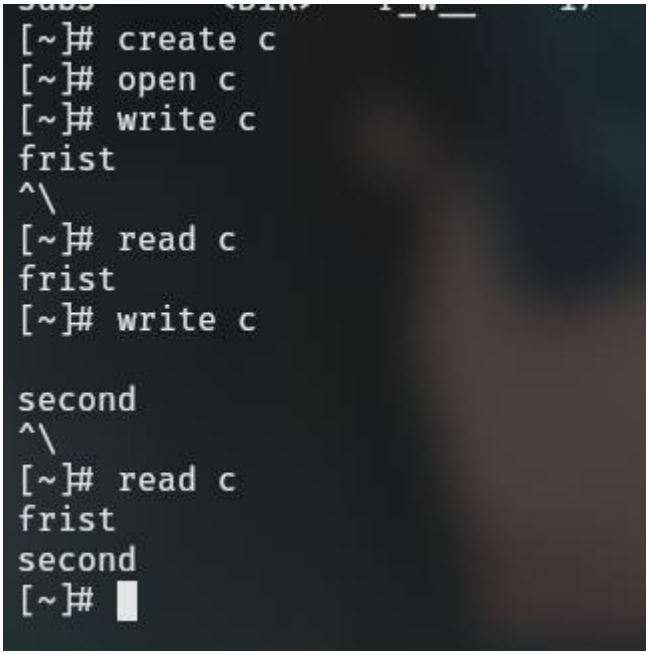
删除文件。

(8)cd



进出不同文件夹。

(9) open close write read



打开文件；向文件中写入；从文件中读出；关闭文件。

(10)overwrite

```
[~]# create c
[~]# open c
[~]# write c
frist
^\
[~]# read c
frist
[~]# write c

second
^\
[~]# read c
frist
second
[~]# cwrite c
clear
^\
[~]# read c
clear
[~]#
```

对文件进行复写。

(11)last access time and last modify time

```
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:20 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:20 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r_w_x    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:03:19 2024  Thu Dec  5 15:02:38 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]# chmod c
change properties to: 4
[~]# open c
[~]# read c
clear
[~]# write c
You do not have permission to write this file.
[~]# close c
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:13 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:13 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r____    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:04:46 2024  Thu Dec  5 15:04:18 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

```
[~]# close c
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:01:30 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:01:30 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r_w_x    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:03:04 2024  Thu Dec  5 15:02:38 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]# open c
[~]# close c
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:05 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:05 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r_w_x    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:03:19 2024  Thu Dec  5 15:02:38 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

```
[~]# format
format acomplished.
Volume Name: EXT2FS
Block Size: 512Bytes
Free Block: 4095
Free Inode: 4095
Directories: 1
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    17          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:11 2024
..        <DIR>   r_w__    17          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:11 2024
[~]# create c.txt
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    29          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:14 2024  Thu Dec  5 15:06:11 2024
..        <DIR>   r_w__    29          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:14 2024  Thu Dec  5 15:06:11 2024
c.txt     <FILE>  r_w__    0           Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024
[~]# create c
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    37          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:42 2024  Thu Dec  5 15:06:11 2024
..        <DIR>   r_w__    37          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:42 2024  Thu Dec  5 15:06:11 2024
c.txt     <FILE>  r_w__    0           Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024
c         <FILE>  r_w_x    0           Thu Dec  5 15:06:56 2024  Thu Dec  5 15:06:56 2024  Thu Dec  5 15:06:56 2024
[~]# mkdir sub
[~]# cd sub
[~/sub]# create a
[~/sub]# create b
[~/sub]# create c.com
[~/sub]# cd ..
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:57 2024  Thu Dec  5 15:06:11 2024
..        <DIR>   r_w__    47          Thu Dec  5 15:06:11 2024  Thu Dec  5 15:06:57 2024  Thu Dec  5 15:06:11 2024
c.txt     <FILE>  r_w__    0           Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024  Thu Dec  5 15:06:40 2024
c         <FILE>  r_w_x    0           Thu Dec  5 15:06:56 2024  Thu Dec  5 15:06:56 2024  Thu Dec  5 15:06:56 2024
sub       <DIR>   r_w__    45          Thu Dec  5 15:07:02 2024  Thu Dec  5 15:07:02 2024  Thu Dec  5 15:07:02 2024
[~]#
```

准确的记录文件最后一次被修改的时间（包括修改文件内容，write；修改文件权限）

准确的记录文件最后一次被访问的时间（close）

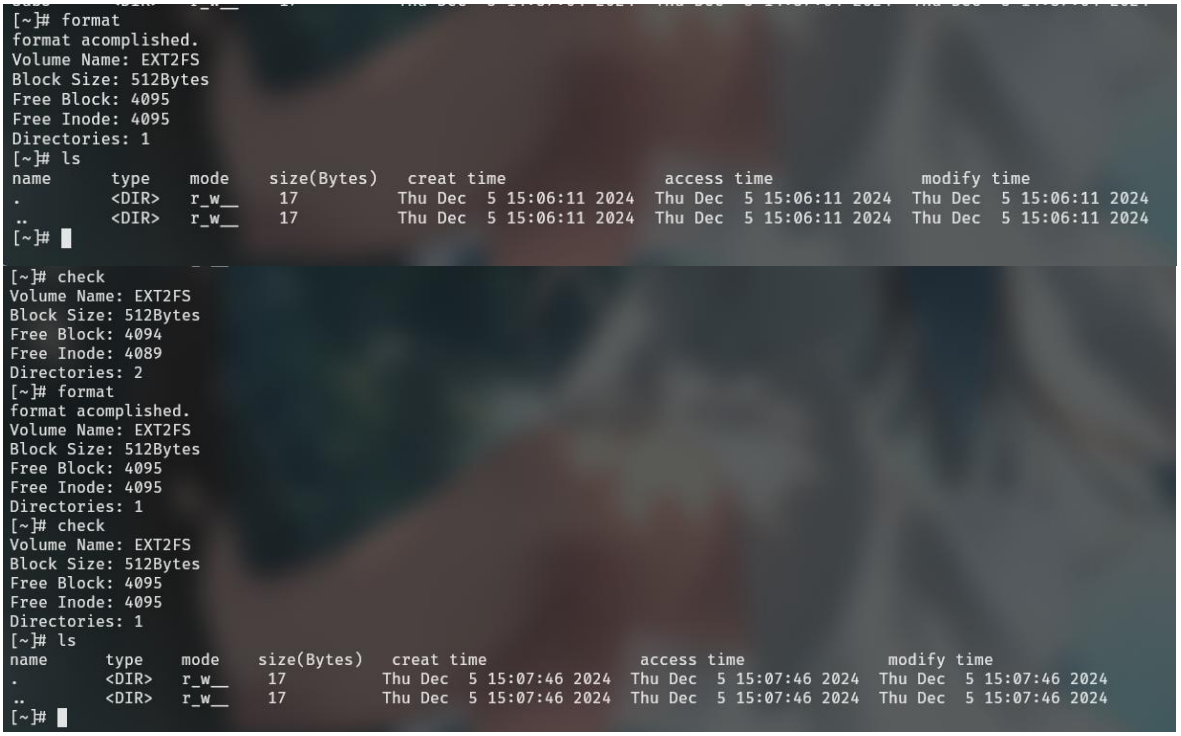
(12)chmod and protection

```
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:20 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:03:20 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r_w_x    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:03:19 2024  Thu Dec  5 15:02:38 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]# chmod c
change properties to: 4
[~]# open c
[~]# read c
clear
[~]# write c
You do not have permission to write this file.
[~]# close c
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:13 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:13 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r____    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:04:46 2024  Thu Dec  5 15:04:18 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

```
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:50 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:04:50 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r____    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:04:46 2024  Thu Dec  5 15:04:18 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]# chmod c
change properties to: 5
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:05:10 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:05:10 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r__x_     6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:05:29 2024  Thu Dec  5 15:05:29 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]# chmod c
change properties to: 7
[~]# ls
name      type    mode    size(Bytes)   creat time                access time               modify time
.         <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:05:30 2024  Thu Dec  5 14:56:55 2024
..        <DIR>   r_w__    47          Thu Dec  5 14:56:55 2024  Thu Dec  5 15:05:30 2024  Thu Dec  5 14:56:55 2024
c         <FILE>  r_w_x    6           Thu Dec  5 15:02:34 2024  Thu Dec  5 15:05:35 2024  Thu Dec  5 15:05:35 2024
sub2      <DIR>   r_w__    17          Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024  Thu Dec  5 14:57:02 2024
sub3      <DIR>   r_w__    17          Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024  Thu Dec  5 14:57:04 2024
[~]#
```

修改文件权限，并对没有相关权限的文件进行保护。

(13)check and format

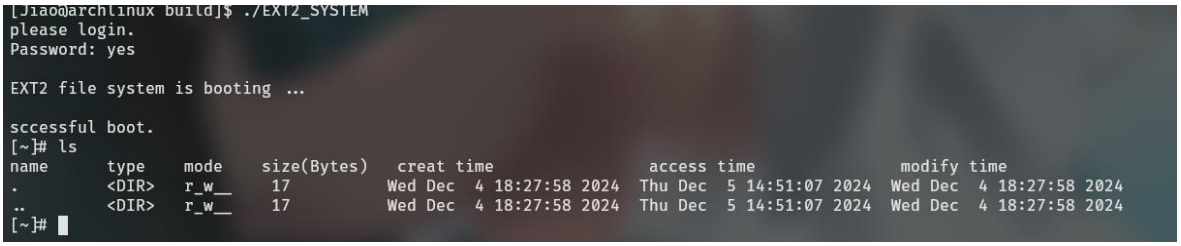


check 可以检查磁盘使用情况

Format 可以格式化磁盘，回到初始状态。

(14)password

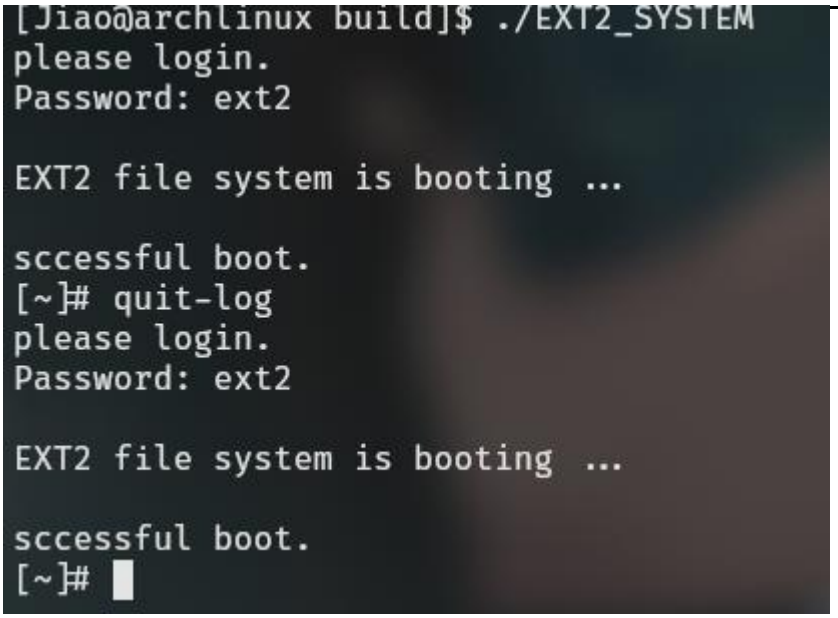
```
[Jiao@archlinux build]$ ./EXT2_SYSTEM
please login.
Password: ext2

EXT2 file system is booting ...

sccessful boot.
[~]# password
Current password: asfds
Password error.
[~]# password
Current password: ext2
New password(no more than 9): yes
Confirm password: yes
Password has changed sccessfully.
[~]# quit-log
please login.
Password: ext2
Error!
Password: yes

EXT2 file system is booting ...

sccessful boot.
[~]#
```



利用 password 命令可以修改密码，即使退出文件系统，下次登陆时也需要输入修改后的新密码。

(15) quit (quit this program) and quit-log`(quit crruent user)

```
[Jiao@archlinux build]$ ./EXT2_SYSTEM
please login.
Password: ext2

EXT2 file system is booting ...

sccessful boot.
[~]# quit
[Jiao@archlinux build]$
```



quit 用以退出文件系统，quit-log 用于退出登录（回到登录界面）。

3.6 实验总结

在本次实验中，我通过手动编程实现了一个类 EXT2 格式的文件系统，可以完成 login, relogin, quit, quit-log, ls, cd, mkdir, rmdir(rm -r), create(touch), delete(rm), open, close, write, read, overwrite, format, check 等等命令。

同时，由于代码过长，我也利用 cmake 和 ninja 工具对代码进行了封装，使得看起来简洁易懂。

3.6.1 实验中的问题与解决过程

问题 1：

cmake 过程中出现的 redeclare 问题。

解决：

通过将基础参数单独定义，定义”basic.h”和”basic.c”文件，防止多重定义。

通过写 #ifndef.....#endif， 防止多重定义。

问题 2：

cmake implicit 问题。

解决方式：

重定向”include/*.h”来完成定义补全。

3.6.2 实验收获

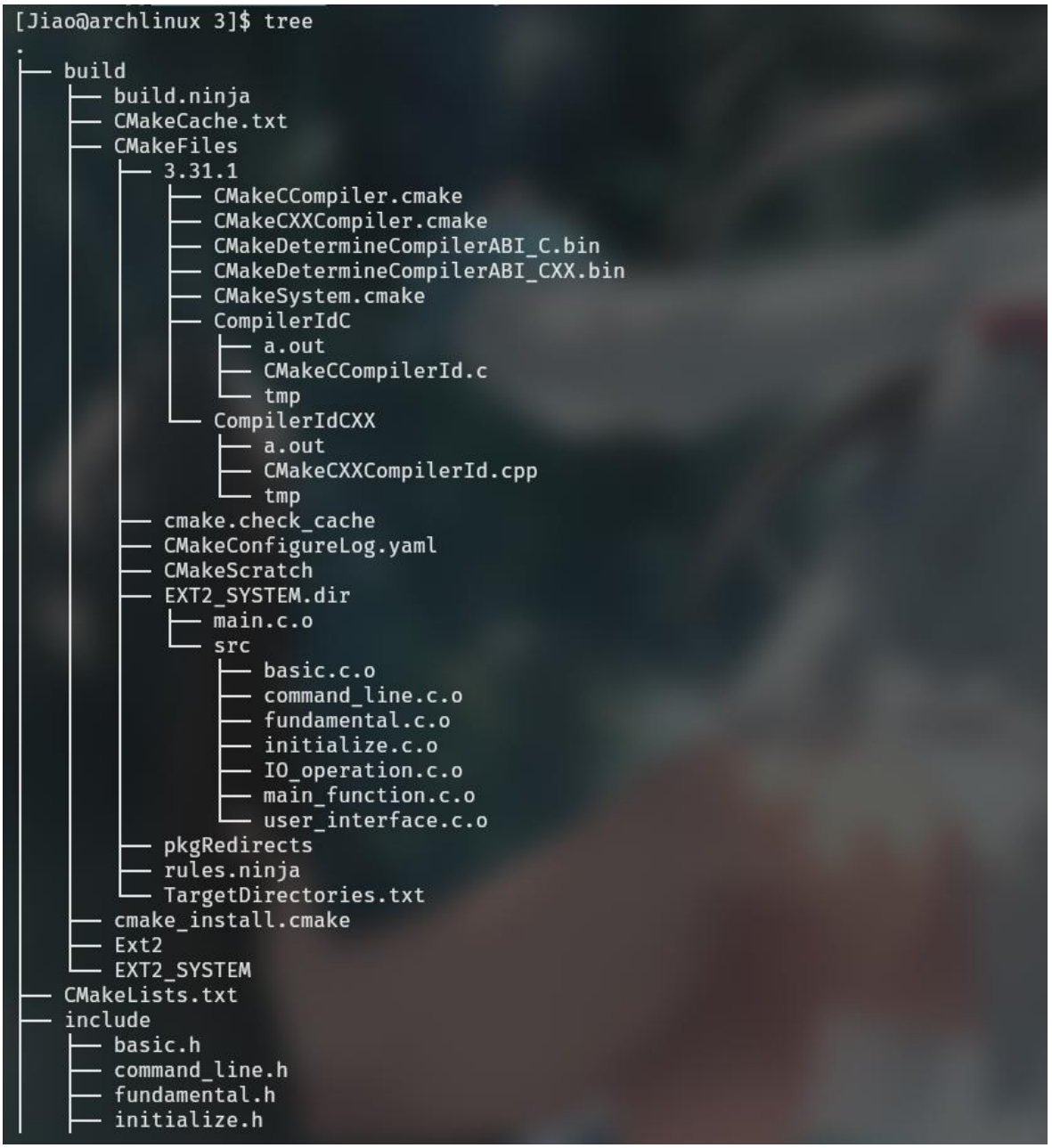
通过本次实验，我深入了解了文件系统的底层原理，对 EXT2 文件系统的设计有了更为直观的认识。在实践过程中，我不仅学习了如何实现文件系统的基本功能，还体验到了复杂代码设计的分层架构方法的重要性。这次实验让我不仅提升了专业技能，还增强了系统性思维与解决问题的能力，受益匪浅。

3.6.3 意见与建议

建议提供视频等相关资料，能够更生动地讲解相关背景知识。

3.7 附件

由于代码过长，这里先展示文件组成架构。



Cmakelists：

cmake_minimum_required(VERSION 3.11)

project(EXT2_SYSTEM VERSION 1.0)

include_directories(${PROJECT_SOURCE_DIR}/include)

```
set(SRC_FILES
    src/command_line.c
    src/initialize.c
    src/fundamental.c
    src/IO_operation.c
    src/user_interface.c
    src/basic.c
    src/main_function.c
    main.c
  )
add_executable(EXT2_SYSTEM ${SRC_FILES})
```

Main.c

```
#include"main_function.h"

int main(void) {
  main_function();
  return 0;
}
```

Basic.h

```
#ifndef GLOBAL_DEFINITIONS_H

#define GLOBAL_DEFINITIONS_H


#include <stdio.h>

#include <signal.h>

#include <string.h>

#include <time.h>


// Constants and macros

#define VOLUME_NAME "EXT2FS"

#define EXT2_NAME_LEN 255

#define FOPEN_TABLE_MAX 16

#define BLOCK_SIZE 512

#define DATA_BLOCK_COUNT 4096

#define INODE_COUNT 4096

#define DISK_SIZE 4611

#define READ_DISK 1

#define WRITE_DISK 0

#define GDT_START 0

#define BLOCK_BITMAP_START 512

#define INODE_BITMAP_START 1024
```

```
#define INODE_TABLE_START 1536

#define DATA_BLOCK_START (1536 + 512 * 512)


// Enumerations

enum {

    FT_UNKNOWN,

    FT_REG_FILE,

    FT_DIR,

    FT_CHRDEV,

    FT_BLKDEV,

    FT_FIFO,

    FT_SOCK,

    FT_SYMLINK,

};


// Struct definitions

typedef struct group_desc {

    char bg_volume_name[10];

    unsigned short bg_block_bitmap;

    unsigned short bg_inode_bitmap;

    unsigned short bg_inode_table;

    unsigned short bg_free_blocks_count;

    unsigned short bg_free_inodes_count;
```

```
    unsigned short bg_used_dirs_count;
    char bg_password[10];
} group_desc;

typedef struct inode {
    unsigned short i_mode;
    unsigned short i_blocks;
    unsigned int i_size;
    unsigned int i_atime;
    unsigned int i_ctime;
    unsigned int i_mtime;
    unsigned int i_dtime;
    unsigned short i_block[8];
    char i_pad[24];
} inode;

typedef struct dir_entry {
    unsigned short inode;
    unsigned short rec_len;
    unsigned char name_len;
    unsigned char file_type;
    char name[EXT2_NAME_LEN];
} dir_entry;
```

```
// Global variables

extern group_desc gdt;

extern inode inode_buf;

extern dir_entry dir_buf;

extern unsigned char block_bitmap[BLOCK_SIZE];

extern unsigned char inode_bitmap[BLOCK_SIZE];

extern unsigned char gdt_Buffer[BLOCK_SIZE];

extern unsigned char inode_Buffer[BLOCK_SIZE];

extern unsigned char Buffer[BLOCK_SIZE * 2];


extern FILE *fp;

extern char current_path[256];

extern unsigned short current_dir;

extern unsigned short fopen_table[FOPEN_TABLE_MAX];

extern char search_file_name[EXT2_NAME_LEN];

extern int flag;


#endif // GLOBAL_DEFINITIONS_H
```

Command_line.h

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void cwrite(char name[]);

unsigned short login(char password[]);

void change_password(void);

void dir(void);

void mkdir(char name[]);

void rmdir(char name[]);
```

```
void create(char name[]);

void delete(char name[]);

void cd(char path[]);

void attrib(char name[], unsigned char change);

void open(char name[]);

void close(char name[]);

void read(char name[]);

void stopWrite();

void write(char name[]);

void check_disk(void);

void format(void);
```

Fundamental.h

```
#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <time.h>


int bitmap_find_0(unsigned char bitmap[]);


void bitmap_neg_k(unsigned char bitmap[], int k);


void initialize_inode(void);


unsigned short new_inode(void);


void free_inode(unsigned short i);
```

```
unsigned short new_block(void);

void free_block(unsigned short i);

void new_dir_entry(char file_name[], unsigned char file_type);

unsigned short is_open(unsigned short inode_num);

unsigned short access_file(unsigned short inode_num,
                           unsigned short (*func)(unsigned short));

unsigned short search_free_dir_in_block(unsigned short block);

unsigned short search_in_block(unsigned short block);

unsigned short search_file(char name[]);

unsigned short delete_in_block(unsigned short block);

unsigned short free_file_block(unsigned short block);

unsigned short print_file(unsigned short block);
```

```
unsigned short print_dir(unsigned short block);

unsigned short add_file_block(unsigned short inode_num);

char *gets_s(char *buffer, int num);
```

Initialize.h

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void initialize_memory(void);

void initialize_disk(void);
```

IO_operation.h

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

```
#include <string.h>

#include <time.h>

void disk_IO(long offset, void *buf, int rw_flag);


void load_group_desc(void);


void update_group_desc(void);


void load_inode_entry(unsigned short k);


void update_inode_entry(unsigned short k);


void load_block_entry(unsigned short i);


void update_block_entry(unsigned short i);


void load_block_bitmap(void);


void update_block_bitmap(void);


void load_inode_bitmap(void);


void update_inode_bitmap(void);
```

Main_function.h

```
#include <stdio.h>

#include <signal.h>

#include <string.h>

#include <time.h>



void main_function(void);
```

User_interface.h

```
#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <time.h>


void shell(void);
```

Basic.c

```
#include"basic.h"


group_desc gdt;
inode inode_buf;
dir_entry dir_buf;
unsigned char block_bitmap[BLOCK_SIZE];
unsigned char inode_bitmap[BLOCK_SIZE];
unsigned char gdt_Buffer[BLOCK_SIZE];
unsigned char inode_Buffer[BLOCK_SIZE];
unsigned char Buffer[BLOCK_SIZE * 2];


FILE *fp = NULL;
char current_path[256];
unsigned short current_dir;
unsigned short fopen_table[FOPEN_TABLE_MAX];
char search_file_name[EXT2_NAME_LEN];
int flag = 1;
```

Command_line.c

```
#include "IO_operation.h"

#include "basic.h"

#include "command_line.h"

#include "fundamental.h"

#include "initialize.h"

#include "user_interface.h"

#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <time.h>



void cwrite(char name[]){

  close(name);

  delete(name);

  create(name);

  open(name);

  write(name);
```

```
}


void stopWrite() { flag = 0; }
void write(char name[]) {
  unsigned short i = search_file(name);
  if (i == 0)
    printf("The file does not exist.\n");
  else {
    load_inode_entry(i);
    unsigned char permission = ((unsigned char *)(&(inode_buf.i_mode)))[1];
    if (permission == 2 || permission == 6 || permission == 7) {
      if (is_open(i)) {

        unsigned short pos = 0, j = 0;
        if (inode_buf.i_blocks == 0) {
          j = new_block();
          inode_buf.i_blocks = 1;
          inode_buf.i_block[0] = j;
          update_inode_entry(i);
          load_block_entry(j);
          pos = 0;
        } else if (inode_buf.i_blocks <= 6) {
          j = inode_buf.i_block[inode_buf.i_blocks - 1];
```

```
  load_block_entry(j);
  pos = inode_buf.i_size % BLOCK_SIZE;
  if (pos == 0) {
    j = add_file_block(i);
    load_block_entry(j);
  }
 } else if (inode_buf.i_blocks == 7) {
  load_block_entry(inode_buf.i_block[6]);
  pos = inode_buf.i_size / BLOCK_SIZE - 6;
  j = ((unsigned short *)Buffer)[pos];
  load_block_entry(j);
  pos = inode_buf.i_size % BLOCK_SIZE;
  if (pos == 0) {
    j = add_file_block(i);
    load_block_entry(j);
  }
 } else {
  load_block_entry(inode_buf.i_block[7]);
  pos = (inode_buf.i_size / BLOCK_SIZE - 6 - 256) / 256;
  load_block_entry(((unsigned short *)Buffer)[pos]);
  pos = (inode_buf.i_size / BLOCK_SIZE - 6 - 256) % 256;
  j = ((unsigned short *)Buffer)[pos];
  load_block_entry(j);
```

```
  pos = inode_buf.i_size % BLOCK_SIZE;
  if (pos == 0) {
    j = add_file_block(i);
    load_block_entry(j);
  }
}

unsigned short new_size = 0;
char ch = getchar();
signal(SIGQUIT, stopWrite);
while (flag /* ch != EOF */) {
  Buffer[pos] = ch;
  pos++;
  new_size++;
  ch = getchar();
  if (pos == BLOCK_SIZE) {
    update_block_entry(j);
    j = add_file_block(i);
    load_block_entry(j);
    pos = 0;
  }
}
flag = 1;
```

```
      Buffer[pos] = '\0';
      update_block_entry(j);
      inode_buf.i_size += new_size - 1;
      inode_buf.i_atime = time(NULL);
      inode_buf.i_mtime = time(NULL);
      update_inode_entry(i);
    } else
      printf("The file does not open.\n");
  } else
    printf("You do not have permission to write this file.\n");
 }
}

void check_disk(void) {
 load_group_desc();
 printf("Volume Name: %s\n", gdt.bg_volume_name);
 printf("Block Size: %dBytes\n", BLOCK_SIZE);
 printf("Free Block: %u\n", gdt.bg_free_blocks_count);
 printf("Free Inode: %u\n", gdt.bg_free_inodes_count);
 printf("Directories: %u\n", gdt.bg_used_dirs_count);
}

void format(void) {
```

```
  initialize_memory();
  initialize_disk();
  printf("format acomplished.\n");
  check_disk();
}

unsigned short login(char password[]) {
  load_group_desc();
  return !(strcmp(gdt.bg_password, password));
}

void change_password(void) {
  printf("Current password: ");
  char password[10];
  gets_s(password, 9);
  if (login(password)) {
    printf("New password(no more than 9): ");
    gets_s(password, 9);
    char confirm[10];
    printf("Confirm password: ");
    gets_s(confirm, 9);
    if (!strcmp(password, confirm)) {
      load_group_desc();
```

```
    strcpy(gdt.bg_password, password);
    update_group_desc();
    printf("Password has changed sccessfully.\n");
   } else {
    printf("The two password not match.\n");
    return;
   }
  } else
   printf("Password error.\n");
}

void dir(void) {
  printf("name      ");
  printf("type    ");
  printf("mode    ");
  printf("size(Bytes)   ");
  printf("creat time              ");
  printf("access time             ");
  printf("modify time             \n");
  access_file(current_dir, print_dir);
  load_inode_entry(current_dir);
  inode_buf.i_atime = time(NULL);
  update_inode_entry(current_dir);
```

```
}

void mkdir(char name[]) {
  unsigned short i = search_file(name);
  if (i) {
    printf("A directory with the same name exists.\n");
  } else {
    new_dir_entry(name, FT_DIR);

    if (!access_file(current_dir, search_free_dir_in_block)) {

      unsigned short j = add_file_block(i);
      memset(Buffer, 0, sizeof(Buffer));
      ((dir_entry *)Buffer)[0] = dir_buf;
      update_block_entry(j);
    }
    load_inode_entry(current_dir);
    inode_buf.i_size += (7 + strlen(name));
    update_inode_entry(current_dir);
  }
}

void rmdir(char name[]) {
```

```
if ((!strcmp(name, ".")) || (!strcmp(name, ".."))) {
  printf("bash: %s :command not found.\n", name);
  return;
}
unsigned short i = search_file(name);
if (i) {
  load_inode_entry(i);
  if (((unsigned char *)(&(inode_buf.i_mode)))[0] != FT_DIR) {
    printf("command not found.\n");
    return;
  }
  //if (inode_buf.i_size != 17) {
  //  printf("Cannot delete non empty directory!\n");
  //  return;
  //}

  access_file(current_dir, delete_in_block);
  access_file(i, free_file_block);
  load_inode_entry(i);
  inode_buf.i_dtime = time(NULL);
  update_inode_entry(i);
  free_inode(i);
```

```
    load_inode_entry(current_dir);
    inode_buf.i_size -= (7 + strlen(name));
    update_inode_entry(current_dir);

    load_group_desc();
    gdt.bg_used_dirs_count--;
    update_group_desc();
  } else {
    printf("The directory does not exist.\n");
  }
}

void create(char name[]) {
  unsigned short i = search_file(name);
  if (i) {
    printf("A file with the same name exists.\n");
  } else {
    new_dir_entry(name, FT_REG_FILE);

    if (!access_file(current_dir, search_free_dir_in_block)) {

      unsigned short j = add_file_block(i);
      memset(Buffer, 0, sizeof(Buffer));
```

```
    ((dir_entry *)Buffer)[0] = dir_buf;
    update_block_entry(j);
   }
   load_inode_entry(current_dir);
   inode_buf.i_size += (7 + strlen(name));
   update_inode_entry(current_dir);
  }
}

void delete(char name[]) {
  unsigned short i = search_file(name);
  if (i) {
   load_inode_entry(i);
   if (((unsigned char *)(&(inode_buf.i_mode)))[0] != FT_REG_FILE) {
    printf("bash : %s : command not found.\n", name);
    return;
   }
   if (is_open(i))
    printf("The file is in use! Please close it first.\n");
   else {
    access_file(current_dir, delete_in_block);

    access_file(i, free_file_block);
```

```
    load_inode_entry(i);
    inode_buf.i_dtime = time(NULL);
    update_inode_entry(i);
    free_inode(i);

    load_inode_entry(current_dir);
    inode_buf.i_size -= (7 + strlen(name));
    update_inode_entry(current_dir);
   }
  } else {
   printf("The file does not exist.\n");
  }
}

void cd(char path[]) {
 if (!strcmp(path, "") || !strcmp(path, "~")) {
   current_dir = 1;
   strcpy(current_path, "root");
  } else if (!strcmp(path, ".")) {
   ;
  } else if (!strcmp(path, "..")) {
   load_inode_entry(current_dir);
   load_block_entry(inode_buf.i_block[0]);
```

```
    current_dir = ((dir_entry *)(Buffer + 8))[0].inode;
    for (int k = strlen(current_path); k >= 0; --k) {
      if (current_path[k] == '/') {
        current_path[k] = '\0';
        break;
      }
    }
  } else {
    unsigned short i = search_file(path);
    if (i) {
      load_inode_entry(i);
      if (((unsigned char *)(&(inode_buf.i_mode)))[0] != FT_DIR) {
        printf("No such directory exist.\n");
        return;
      }
      current_dir = i;
      strcat(current_path, "/");
      strcat(current_path, path);
    } else {
      printf("No such directory exist.\n");
    }
  }
}
```

```
void attrib(char name[], unsigned char change) {
  unsigned short i = search_file(name);
  if (i == 0)
    printf("The file does not exist.\n");
  else {
    if (change == 2 || change == 4 || change == 6 || change == 7 ||change == 5 ||
change ==1) {
      load_inode_entry(i);
      ((unsigned char *)(&(inode_buf.i_mode)))[1] = change;
      inode_buf.i_atime = time(NULL);
      inode_buf.i_mtime = time(NULL);
      update_inode_entry(i);
    } else
      printf("illigal change.\n");
  }
}

void open(char name[]) {
  unsigned short inode_num = search_file(name);
  if (inode_num == 0)
    printf("The file does not exist.\n");
  else {
    if (is_open(inode_num))
```

```
    printf("The file has opened.\n");
   else {
    load_inode_entry(inode_num);
    unsigned char permission = ((unsigned char *)(&(inode_buf.i_mode)))[1];
    if (permission == 4 || permission == 6 || permission == 7) {
     inode_buf.i_atime = time(NULL);
     update_inode_entry(inode_num);
     for (unsigned short i = 0; i < FOPEN_TABLE_MAX; ++i) {
      if (fopen_table[i] == 0) {
       fopen_table[i] = inode_num;
       return;
      }
     }
     printf("The number of files opened has reached the maximum.\n");
    } else
     printf("You do not have permission to open this file.\n");
   }
  }
}

void close(char name[]) {
  unsigned short inode_num = search_file(name);
  if (inode_num == 0)
```

```
  printf("The file does not exist.\n");
 else {
  if (is_open(inode_num)) {
   load_inode_entry(inode_num);
   inode_buf.i_atime = time(NULL);
   update_inode_entry(inode_num);
   for (unsigned short i = 0; i < FOPEN_TABLE_MAX; ++i) {
    if (fopen_table[i] == inode_num) {
     fopen_table[i] = 0;
     return;
    }
   }
  } else
   printf("The file does not open.\n");
 }
}

void read(char name[]) {
 unsigned short i = search_file(name);
 if (i == 0)
  printf("The file does not exist.\n");
 else {
  load_inode_entry(i);
```

```
    unsigned char permission = ((unsigned char *)(&(inode_buf.i_mode)))[1];
    if (permission == 4 || permission == 6 || permission == 7) {
      if (is_open(i)) {
        access_file(i, print_file);
        inode_buf.i_atime = time(NULL);
        update_inode_entry(i);
      } else
        printf("The file does not open.\n");
    } else
      printf("You do not have permission to read this file.\n");
  }
}
```

Fundamental.c

```
#include "IO_operation.h"
#include "basic.h"
#include "command_line.h"
#include "fundamental.h"
#include "initialize.h"
#include <signal.h>
```

```
#include <stdio.h>

#include <string.h>

#include <time.h>


int bitmap_find_0(unsigned char bitmap[]) {

  for (int i = 0; i < BLOCK_SIZE; ++i) {

    unsigned char mask = 0b10000000;

    if (bitmap[i] == 0b11111111)

      continue;

    for (int j = 0; j < 8; ++j) {

      if (!(bitmap[i] & mask))

        return i * 8 + j;

      mask >>= 1;

    }

  }

  return -1;

}


void bitmap_neg_k(unsigned char bitmap[], int k) {

  int i = k / 8, j = k % 8;

  unsigned char mask = 0b10000000;

  for (int t = 0; t < j; ++t)

    mask >>= 1;
```

```
  bitmap[i] ^= mask;
}

void initialize_inode(void) {
  inode_buf.i_mode = FT_UNKNOWN;
  inode_buf.i_mode <<= 8;

  ((unsigned char *)(&(inode_buf.i_mode)))[1] = 0b00000110;
  inode_buf.i_blocks = 0;
  inode_buf.i_size = 0;
  inode_buf.i_atime = time(NULL);
  inode_buf.i_ctime = time(NULL);
  inode_buf.i_mtime = time(NULL);
  inode_buf.i_dtime = 0;
  memset(inode_buf.i_block, 0, sizeof(inode_buf.i_block));
  memset(inode_buf.i_pad, 0xff, sizeof(inode_buf.i_pad));
}

unsigned short new_inode(void) {
  load_group_desc();
  if (gdt.bg_free_inodes_count) {

    load_inode_bitmap();
```

```
    unsigned short i = bitmap_find_0(inode_bitmap);
    bitmap_neg_k(inode_bitmap, i);
    update_inode_bitmap();

    initialize_inode();
    update_inode_entry(i + 1);

    gdt.bg_free_inodes_count--;
    update_group_desc();
    return i + 1;
  } else {
    printf("There is no inode to be allocated!\n");
    return 0;
  }
}

void free_inode(unsigned short i) {

  load_inode_bitmap();
  bitmap_neg_k(inode_bitmap, i - 1);
  update_inode_bitmap();

  load_group_desc();
```

```
  gdt.bg_free_inodes_count++;
  update_group_desc();
}

unsigned short new_block(void) {
  load_group_desc();
  if (gdt.bg_free_blocks_count) {

    load_block_bitmap();
    unsigned short i = bitmap_find_0(block_bitmap);
    bitmap_neg_k(block_bitmap, i);
    update_block_bitmap();

    gdt.bg_free_blocks_count--;
    update_group_desc();
    return i;
  } else {
    printf("There is no block to be allocated!\n");
    return 0;
  }
}

void free_block(unsigned short i) {
```

```
  load_block_bitmap();
  bitmap_neg_k(block_bitmap, i);
  update_block_bitmap();

  load_group_desc();
  gdt.bg_free_blocks_count++;
  update_group_desc();
}

void new_dir_entry(char file_name[], unsigned char file_type) {
  memset(&dir_buf, 0, sizeof(dir_entry));
  dir_buf.inode = new_inode();
  dir_buf.name_len = strlen(file_name);
  dir_buf.rec_len = 7 + dir_buf.name_len;
  dir_buf.file_type = file_type;
  strcpy(dir_buf.name, file_name);

  load_inode_entry(dir_buf.inode);
  ((unsigned char *)(&(inode_buf.i_mode)))[0] = file_type;
  char *extension = strchr(file_name, '.');
  if (!extension || !strcmp(extension, ".exe") || !strcmp(extension, ".bin") ||
     !strcmp(extension, ".com"))
```

```
 ((unsigned char *)(&(inode_buf.i_mode)))[1] = 0b00000111;

update_inode_entry(dir_buf.inode);


if (file_type == FT_DIR) {


 load_group_desc();

 gdt.bg_used_dirs_count++;

 update_group_desc();


 unsigned short i = new_block();

 memset(Buffer, 0, sizeof(Buffer));

 dir_entry temp;


 memset(&temp, 0, sizeof(dir_entry));

 temp.inode = dir_buf.inode;

 temp.rec_len = 8;

 temp.name_len = 1;

 temp.file_type = FT_DIR;

 strcpy(temp.name, ".");

 ((dir_entry *)Buffer)[0] = temp;


 memset(&temp, 0, sizeof(dir_entry));

 temp.inode = current_dir;
```

```
    temp.rec_len = 9;
    temp.name_len = 2;
    temp.file_type = FT_DIR;
    strcpy(temp.name, "..");
    ((dir_entry *)(Buffer + 8))[0] = temp;

    update_block_entry(i);

    load_inode_entry(dir_buf.inode);
    inode_buf.i_blocks = 1;
    inode_buf.i_size = 17;
    inode_buf.i_block[0] = i;
    ((unsigned char *)(&(inode_buf.i_mode)))[1] = 0b00000110;
    update_inode_entry(dir_buf.inode);
  }
}

unsigned short is_open(unsigned short inode_num) {
  for (int i = 0; i < FOPEN_TABLE_MAX; ++i)
    if (fopen_table[i] == inode_num)
      return 1;
  return 0;
}
```

```
unsigned short access_file(unsigned short inode_num,
                           unsigned short (*func)(unsigned short)) {

 load_inode_entry(inode_num);
 unsigned short indirect_1 = 0, indirect_2 = 0;
 for (unsigned short i = 0; i < inode_buf.i_blocks; /**/) {
  if (i < 6) {

    load_block_entry(inode_buf.i_block[i]);
    unsigned short ret = func(inode_buf.i_block[i]);
    update_block_entry(inode_buf.i_block[i]);
    ++i;
    if (ret)
      return ret;
  } else if (i == 6) {

    load_block_entry(inode_buf.i_block[i]);
    unsigned short j;
    if (indirect_1 < BLOCK_SIZE / sizeof(unsigned short))
      j = ((unsigned short *)Buffer)[indirect_1];
    if (j == 0 || indirect_1 == BLOCK_SIZE / sizeof(unsigned short)) {
      ++i;
```

```
    indirect_1 = 0;
    continue;
  }
  ++indirect_1;
  load_block_entry(j);
  unsigned short ret = func(j);
  update_block_entry(j);
  if (ret)
    return ret;
 } else {

  load_block_entry(inode_buf.i_block[i]);
  unsigned short j, k;
  if (indirect_1 < BLOCK_SIZE / sizeof(unsigned short))
    j = ((unsigned short *)Buffer)[indirect_1];
  if (j == 0 || indirect_1 == BLOCK_SIZE / sizeof(unsigned short))
    break;
  load_block_entry(j);
  if (indirect_2 < BLOCK_SIZE / sizeof(unsigned short))
    k = ((unsigned short *)Buffer)[indirect_2];
  if (k == 0)
    break;
  if (indirect_2 == BLOCK_SIZE / sizeof(unsigned short)) {
```

```
      ++indirect_1;
      indirect_2 = 0;
      continue;
    }
    ++indirect_2;
    load_block_entry(k);
    unsigned short ret = func(k);
    update_block_entry(k);
    if (ret)
      return ret;
   }
  }
  return 0;
}

unsigned short search_free_dir_in_block(unsigned short block) {
  unsigned short current_pos = 0;

  memset(Buffer + BLOCK_SIZE, 0, BLOCK_SIZE);
  dir_entry temp = ((dir_entry *)Buffer)[0];
  do {
    unsigned short k = temp.rec_len - temp.name_len - 7;
    if (k >= dir_buf.rec_len) {
```

```
    temp.rec_len = temp.name_len + 7;
    ((dir_entry *)(Buffer + current_pos))[0].rec_len = temp.rec_len;
    dir_buf.rec_len = k;
    ((dir_entry *)(Buffer + current_pos + temp.rec_len))[0].inode =
       dir_buf.inode;
    ((dir_entry *)(Buffer + current_pos + temp.rec_len))[0].rec_len =
       dir_buf.rec_len;
    ((dir_entry *)(Buffer + current_pos + temp.rec_len))[0].name_len =
       dir_buf.name_len;
    ((dir_entry *)(Buffer + current_pos + temp.rec_len))[0].file_type =
       dir_buf.file_type;
    strcpy(((dir_entry *)(Buffer + current_pos + temp.rec_len))[0].name,
          dir_buf.name);
    return 1;
  }
  current_pos += temp.rec_len;
  temp = ((dir_entry *)(Buffer + current_pos))[0];
} while (temp.inode);
if (BLOCK_SIZE - current_pos > dir_buf.rec_len) {
  ((dir_entry *)(Buffer + current_pos))[0] = dir_buf;
  return 1;
}
return 0;
```

```
}

unsigned short search_in_block(unsigned short block) {
  unsigned short current_pos = 0;
  memset(Buffer + BLOCK_SIZE, 0, BLOCK_SIZE);
  dir_buf = ((dir_entry *)Buffer)[0];
  do {
    if (!strcmp(dir_buf.name, search_file_name))
      return dir_buf.inode;
    current_pos += dir_buf.rec_len;
    dir_buf = ((dir_entry *)(Buffer + current_pos))[0];
  } while (dir_buf.inode);
  return 0;
}

unsigned short search_file(char name[]) {
  strcpy(search_file_name, name);
  return access_file(current_dir, search_in_block);
}

unsigned short delete_in_block(unsigned short block) {
  unsigned short current_pos = 0, pre_pos = 0;
```

```
  memset(Buffer + BLOCK_SIZE, 0, BLOCK_SIZE);
  dir_buf = ((dir_entry *)Buffer)[0];
  do {
    if (!strcmp(dir_buf.name, search_file_name)) {
      ((dir_entry *)(Buffer + pre_pos))[0].rec_len += dir_buf.rec_len;
      return dir_buf.inode;
    }
    pre_pos = current_pos;
    current_pos += dir_buf.rec_len;
    dir_buf = ((dir_entry *)(Buffer + current_pos))[0];
  } while (dir_buf.inode);
  return 0;
}

unsigned short free_file_block(unsigned short block) {
  free_block(block);
  return 0;
}

unsigned short print_file(unsigned short block) {
  for (unsigned short i = 0; i < BLOCK_SIZE; ++i) {
    if (Buffer[i] == 0)
      return 1;
```

```
    putchar(Buffer[i]);
  }
  return 0;
}

unsigned short print_dir(unsigned short block) {
  inode inode_temp = inode_buf;
  unsigned short current_pos = 0;

  memset(Buffer + BLOCK_SIZE, 0, BLOCK_SIZE);
  dir_buf = ((dir_entry *)Buffer)[0];
  do {
    load_inode_entry(dir_buf.inode);
    printf("%-10s", dir_buf.name);
    switch (dir_buf.file_type) {
    case FT_DIR:
      printf("<DIR>    ");
      break;
    case FT_REG_FILE:
      printf("<FILE>   ");
      break;
    default:
      printf("unknown ");
```

```
}
switch (((unsigned char *)(&(inode_buf.i_mode)))[1]) {
case 2:
  printf("__w__    ");
  break;
case 5:
  printf("r__x_    ");
  break;
case 4:
  printf("r___    ");
  break;
case 6:
  printf("r_w__    ");
  break;
case 7:
  printf("r_w_x    ");
  break;
case 1:
  printf("___x_    ");
default:
  printf("error  ");
}
printf("%-12hu", inode_buf.i_size);
```

```
  time_t temp;
  char time_str[26];
  temp = inode_buf.i_ctime;
  strcpy(time_str, ctime(&temp));
  time_str[24] = '\0';
  printf("%s  ", time_str);
  temp = inode_buf.i_atime;
  strcpy(time_str, ctime(&temp));
  time_str[24] = '\0';
  printf("%s  ", time_str);
  temp = inode_buf.i_mtime;
  strcpy(time_str, ctime(&temp));
  time_str[24] = '\0';
  printf("%s  \n", time_str);
  current_pos += dir_buf.rec_len;
  dir_buf = ((dir_entry *)(Buffer + current_pos))[0];
 } while (dir_buf.inode);
 inode_buf = inode_temp;
 return 0;
}

unsigned short add_file_block(unsigned short inode_num) {
 unsigned short i = 0;
```

```
load_inode_entry(inode_num);
if (inode_buf.i_blocks < 6) {
  i = new_block();
  inode_buf.i_blocks++;
  inode_buf.i_block[inode_buf.i_blocks - 1] = i;
} else if (inode_buf.i_blocks == 6) {
  i = new_block();
  inode_buf.i_blocks++;
  inode_buf.i_block[inode_buf.i_blocks - 1] = i;
  i = new_block();
  memset(Buffer, 0, sizeof(Buffer));
  ((unsigned short *)Buffer)[0] = i;
  update_block_entry(inode_buf.i_block[inode_buf.i_blocks - 1]);
} else if (inode_buf.i_blocks == 7) {
  load_block_entry(inode_buf.i_block[6]);
  unsigned short j;
  for (j = 0; j < 256 && ((unsigned short *)Buffer)[j]; ++j)
    ;
  if (j == 256) {
    i = new_block();
    inode_buf.i_blocks++;
    inode_buf.i_block[inode_buf.i_blocks - 1] = i;
    i = new_block();
```

```
    memset(Buffer, 0, sizeof(Buffer));
    ((unsigned short *)Buffer)[0] = i;
    update_block_entry(inode_buf.i_block[inode_buf.i_blocks - 1]);
    unsigned short k = new_block();
    memset(Buffer, 0, sizeof(Buffer));
    ((unsigned short *)Buffer)[0] = k;
    update_block_entry(i);
    i = k;
  } else {
    i = new_block();
    ((unsigned short *)Buffer)[j] = i;
  }
  update_block_entry(inode_buf.i_block[6]);
} else {
  load_block_entry(inode_buf.i_block[7]);
  unsigned short j, k;
  for (j = 0; j < 256 && ((unsigned short *)Buffer)[j]; ++j)
    ;
  --j;
  load_block_entry(j);
  for (k = 0; k < 256 && ((unsigned short *)Buffer)[k]; ++k)
    ;
  if (k < 256) {
```

```
    i = new_block();
    ((unsigned short *)Buffer)[k] = i;
    update_block_entry(j);
   } else {
    if (j < 255) {
     load_block_entry(inode_buf.i_block[7]);
     ++j;
     unsigned short temp = new_block();
     ((unsigned short *)Buffer)[j] = temp;
     update_block_entry(inode_buf.i_block[7]);
     memset(Buffer, 0, sizeof(Buffer));
     i = new_block();
     ((unsigned short *)Buffer)[0] = i;
     update_block_entry(temp);
    } else
     printf("The file has reached the maximum capacity!\n");
   }
  }
  update_inode_entry(inode_num);
  return i;
}

char *gets_s(char *buffer, int num) {
```

```
 if (fgets(buffer, num, stdin) != 0) {
   size_t len = strlen(buffer);
   if (len > 0 && buffer[len - 1] == '\n')
     buffer[len - 1] = '\0';
   return buffer;
 }
 return NULL;
}
```

Initialize.c

```
#include "IO_operation.h"
#include"basic.h"
#include "command_line.h"
#include "fundamental.h"
#include "initialize.h"
#include "user_interface.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void initialize_memory(void) {
```

```
  memset(fopen_table, 0, sizeof(fopen_table));


  strcpy(gdt.bg_volume_name, VOLUME_NAME);
  strcpy(gdt.bg_password, "ext2");
  gdt.bg_block_bitmap = 1;
  gdt.bg_inode_bitmap = 2;
  gdt.bg_inode_table = 3;
  gdt.bg_free_blocks_count = DATA_BLOCK_COUNT;
  gdt.bg_free_inodes_count = INODE_COUNT;
  gdt.bg_used_dirs_count = 0;
  current_dir = 1;
  strcpy(current_path, "~");
}


void initialize_disk(void) {
  memset(Buffer, 0, sizeof(Buffer));
  for (int i = 0; i < DISK_SIZE; ++i) {
    disk_IO(i * BLOCK_SIZE, Buffer, WRITE_DISK);
  }
  update_group_desc();


  new_dir_entry("root", FT_DIR);
```

```
}
```

IO_operation.c

```
#include "IO_operation.h"
#include "basic.h"
#include "command_line.h"
#include "fundamental.h"
#include "initialize.h"
#include "user_interface.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void disk_IO(long offset, void *buf, int rw_flag) {
  fseek(fp, offset, SEEK_SET);
  if (rw_flag) {
    fread(buf, BLOCK_SIZE, 1, fp);
  } else {
    fwrite(buf, BLOCK_SIZE, 1, fp);
    fflush(fp);
```

```
  }
}

void load_group_desc(void) {
  disk_IO(GDT_START, gdt_Buffer, READ_DISK);
  gdt = ((group_desc *)gdt_Buffer)[0];
}

void update_group_desc(void) {
  memset(gdt_Buffer, 0xff, sizeof(gdt_Buffer));
  ((group_desc *)gdt_Buffer)[0] = gdt;
  disk_IO(GDT_START, gdt_Buffer, WRITE_DISK);
}

void load_inode_entry(unsigned short k) {
  --k;
  unsigned short i = k / 8, j = k % 8;
  disk_IO(INODE_TABLE_START + i * BLOCK_SIZE, inode_Buffer, READ_DISK);
  inode_buf = ((inode *)inode_Buffer)[j];
}

void update_inode_entry(unsigned short k) {
  --k;
```

```
  unsigned short i = k / 8, j = k % 8;
  disk_IO(INODE_TABLE_START + i * BLOCK_SIZE, inode_Buffer, READ_DISK);
  ((inode *)inode_Buffer)[j] = inode_buf;
  disk_IO(INODE_TABLE_START + i * BLOCK_SIZE, inode_Buffer, WRITE_DISK);
}

void load_block_entry(unsigned short i) {
  disk_IO(DATA_BLOCK_START + i * BLOCK_SIZE, Buffer, READ_DISK);
}

void update_block_entry(unsigned short i) {
  disk_IO(DATA_BLOCK_START + i * BLOCK_SIZE, Buffer, WRITE_DISK);
}

void load_block_bitmap(void) {
  disk_IO(BLOCK_BITMAP_START, block_bitmap, READ_DISK);
}

void update_block_bitmap(void) {
  disk_IO(BLOCK_BITMAP_START, block_bitmap, WRITE_DISK);
}

void load_inode_bitmap(void) {
```

```
  disk_IO(INODE_BITMAP_START, inode_bitmap, READ_DISK);
}

void update_inode_bitmap(void) {
  disk_IO(INODE_BITMAP_START, inode_bitmap, WRITE_DISK);
}
```

Main_function.c

```
#include "IO_operation.h"
#include "basic.h"
#include "command_line.h"
#include "fundamental.h"
#include "initialize.h"
#include "main_function.h"
#include "user_interface.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void main_function(void) {
```

```
fp = fopen("./Ext2", "rb+");


initialize_memory();


if (fp == NULL) {
  fp = fopen("./Ext2", "wb+");
  initialize_disk();
}
printf("please login. \n");
printf("Password: ");
char password[10];
gets_s(password, 9);
while (!login(password)) {
  printf("Error!\n");
  printf("Password: ");
  gets_s(password, 9);
}


printf("\nEXT2 file system is booting ...\n");
printf("\nsccessful boot.\n");
shell();
fclose(fp);
```

```
  return;
}
```

User_interface.c

```
#include "IO_operation.h"
#include "basic.h"
#include "command_line.h"
#include "fundamental.h"
#include "initialize.h"
#include "user_interface.h"
#include"main_function.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

void shell(void) {
  char cmd[256] = "";
  while (1) {
```

```
printf("[%s]# ", current_path);
gets_s(cmd, 256);
if (!strcmp(cmd, "format")) {
  format();
} else if (!strcmp(cmd, "check")) {
  check_disk();
} else if (!strcmp(cmd, "password")) {
  change_password();
} else if (!strcmp(cmd, "ls")) {
  dir();
} else if (!strncmp(cmd, "mkdir ", 6)) {
  mkdir(cmd + 6);
} else if (!strncmp(cmd, "rmdir ", 6)) {
  rmdir(cmd + 6);
} else if (!strncmp(cmd, "create ", 7)) {
  create(cmd + 7);
} else if (!strncmp(cmd, "delete ", 7)) {
  delete (cmd + 7);
} else if (!strncmp(cmd, "cd ", 3)) {
  cd(cmd + 3);
} else if (!strncmp(cmd, "cwrite ", 7)) {
  cwrite(cmd + 7);
} else if (!strncmp(cmd, "chmod ", 6)) {
```

```
  printf("change properties to: ");
  unsigned char change;
  scanf("%hhu", &change);
  getchar();
  attrib(cmd + 6, change);
} else if (!strncmp(cmd, "open ", 5)) {
  open(cmd + 5);
} else if (!strncmp(cmd, "close ", 6)) {
  close(cmd + 6);
} else if (!strncmp(cmd, "read ", 5)) {
  read(cmd + 5);
} else if (!strncmp(cmd, "write ", 6)) {
  write(cmd + 6);
} else if(!strcmp(cmd, "quit-log")){
  fclose(fp);
  main_function();
}


  else if (!strcmp(cmd, "quit")) {
  break;
} else {
  printf("bash : %s : command not found.\n", cmd);
}
```

```
    }

}
```

3.7.2 附件 2

readme 附于文件最后。