实　验　报　告

学　　号	2103003 1009	姓　　名	惠欣宇	专业班级	计算机 4 班
课程名称	**操作系统**			学期	**2023 年秋季学期**
任课教师	窦金凤	完成日期	2023.12.15	上机课时间	周五 3-6
实 验 名 称	Linux 进程控制—lockf				

一、实验目的及要求

1.了解进程与程序的区别，加深对进程概念的理解；

2.进一步认识进程并发执行的原理，理解进程并发执行的特点，区别进程并发执行与顺序执行；

3.分析进程争用临界资源的现象，学习解决进程互斥的方法。

4.了解 fork()系统调用的返回值，掌握用 fork()创建进程的方法；

5.熟悉 wait、exit 等系统调用。

二、实验内容

利用系统调用 lockf（fd，mode，size），对指定区域（有 size 指示）进行加锁或解锁，以实现进程的同步或互斥。其中，fd 是文件描述字；mode 是锁定方式，mode=1 表示加锁，mode=0 表示解锁；size 是指定文件 fd 的指定区域，用 0 表示从当前位置到文件结尾（注：有些 Linux 系统是 locking（fd，mode，size））

三、实验源码

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void) {
    int pid;
    if((pid = fork()) < 0) {  //创建子进程失败
        printf("child fail create\n");
        return 0;
    } else if (pid == 0) {
        lockf(1, 1, 0);  //锁定标准输出设备(显示器)。只允许子进程使用显示器资源
        //如果不进行锁定，则由于并发执行的不可在现性，子进程与父进程都会去抢占显示器资源
        //从而导致父子进程的输出混乱
        for(int i = 0; i < 100; i++)
            printf("I am child (pid = %d) process.\n",getpid());
        lockf(1, 0, 0);  //解锁标准输出设备(显示器)。把显示器资源释放给父进程
        return 0;
    } else {
        lockf(1, 1, 0);  //同上。这里将显示器资源锁定给父进程
        for(int i = 0; i < 100; i++)
            printf("I am father process.\n");
        lockf(1,0,0);  //同上。将显示器资源释放给子进程
    }
    usleep(5000);
    return 0;
}
```

四、实验结果

下面给出给资源上锁与不上锁的两种情况的输出，如图 1、2。

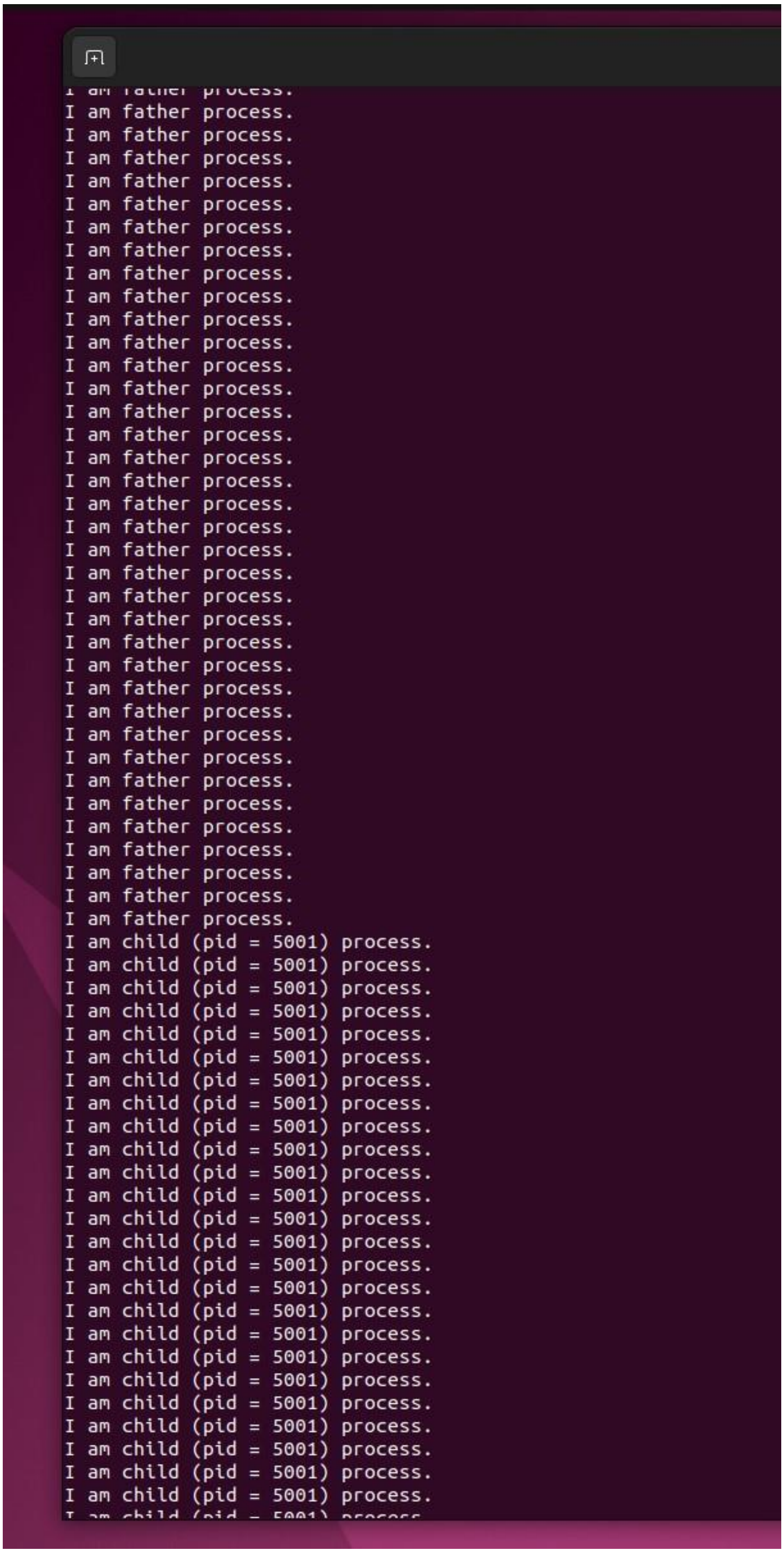

图 1 使用 lockf 函数为资源上锁

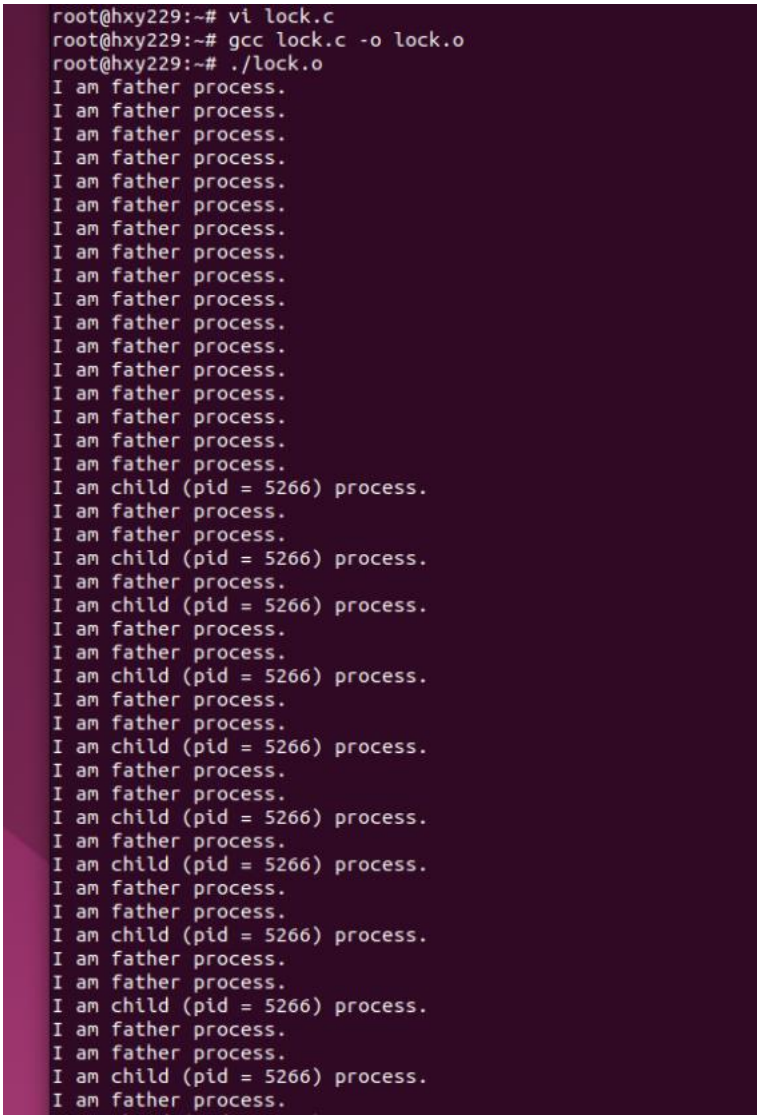

图 2 不使用 lockf 函数为资源上锁

五、结果分析

利用系统调用 lockf（fd，mode，size），对指定区域（有 size 指示）进行加锁或解锁，以实现进程的同步或互斥。其中，fd 是文件描述字；mode 是锁定方式，mode=1 表示加锁，mode=0 表示解锁；size 是指定文件 fd 的指定区域，用 0 表示从当前位置到文件结尾（注：有些 Linux 系统是 locking（fd，mode，size））。

针对输出的实验结果结合实验一的分析，我们可以总结出：父进程创建了紫禁城之后，两个进程都需要执行 fork 之后的代码部分。父进程进入打印“I am father process!”信息部分的代码，子进程进入输出“I am child process！”信息部分的代码。

而在此时便会发生实验一中的情况，即父子进程抢占资源，该资源就是显示器，因为

父子进程并发执行，都需要抢占显示器来执行自己部分的代码内容（输出自己代码部分的信息在显示器上）。但因为此时资源未受限制且父子进程都会循环输出 100 次自己的信息，因此在某一次进程循环打印完成之后，显示器会空闲一小会儿，此时父子进程都有机会去抢占该资源。这个时候哪一个进程抢占成功，便由哪一个进程来执行自己部分的代码，执行输出内容。

我们再次执行未加 lockf 函数的代码，可以看出每一次的输出顺序都不相同，这一结果同样体现了进程并发执行的不可再现性。再次执行的结果如图 3，与上述图 2 做对比。

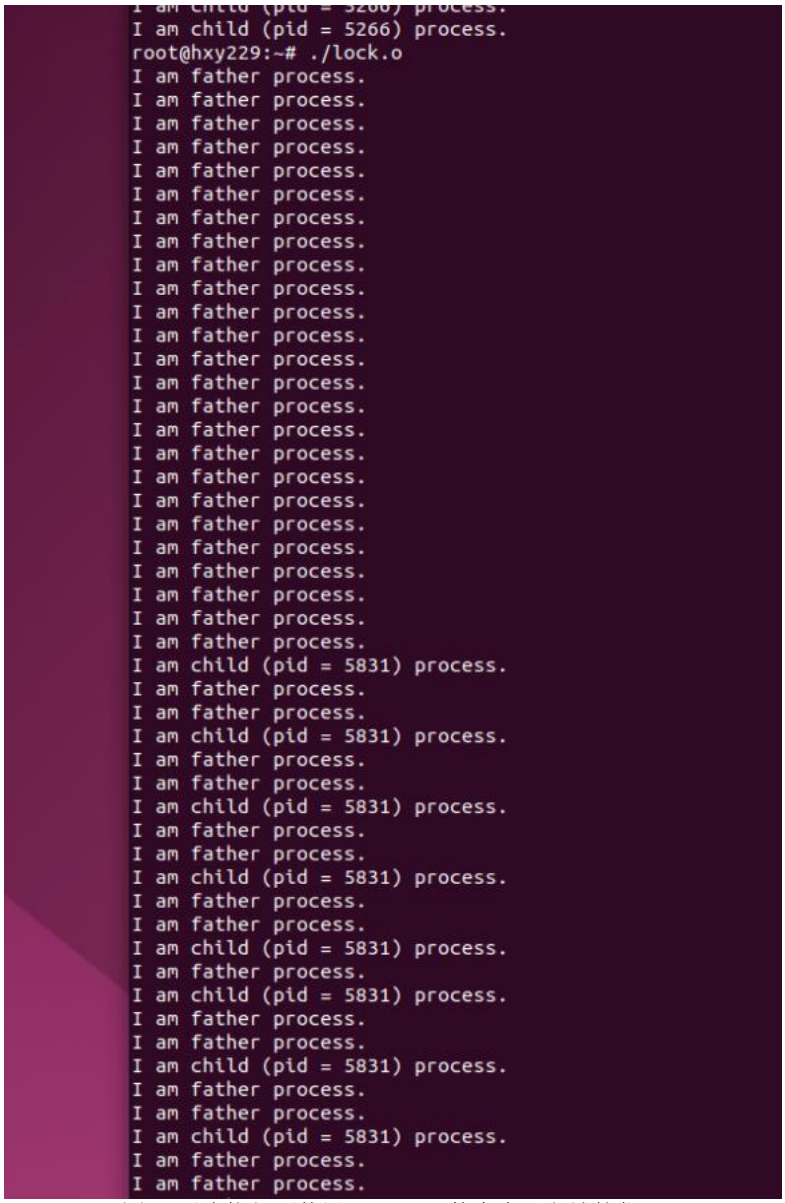

图 3 再次执行不使用 lockf 函数为资源上锁的代码

上面说明了进程执行时资源不上锁的情况，但这必然会带来每次执行的不同结果，我们如果需要每次执行结果都相同，或者说我们不希望让父子进程输出的乱序现象发生。此

时我们就需要使用 lockf 函数对进程所需要的资源进程上锁。比如说，在父子进程抢占显示器资源时，父进程先抢占到了显示器资源，那么父进程会使用 lockf 对显示器资源进行上锁，上锁期间只有父进程可以使用该资源，在父进程执行完自己的所有打印操作之后，才会释放显示器资源。由于父进程已经释放了显示器资源，因此此时子进程可以抢占到显示器资源来执行自己的代码部分（也就是打印自己的输出信息），在子进程中也对显示器资源进行了上锁，保证整个输出过程中不会被其他进程抢占显示器资源。

通过上述对资源上锁的过程即可以实现父子进程分别对显示器资源的独占，且输出不会出现乱序，虽然此时进程还是并发执行，但由于对资源的上锁，便不会出现不可再现性。也是一次很有意思的实验。