**海南大学网络空间学院**

**实 验 报 告**

**实验课程： 《操作系统原理及安全》实验**

**实验名称：** 实验四

**学号：** 20213006839  **姓名：** 甄五四

**专业班级： 信息安全（密码学方向）理科实验班指导教师：** 秦小立

**完成日期：** 2023 年 5 月 24 日

**成绩：**

**评阅：**

**教师签名：**

1. **实验目的**

1、进一步认识并发执行的实质

2、分析进程竞争资源的现象，学习解决进程同步互斥的方法

1. **实验任务**

1、编写程序，使用相关函数实现父子进程对共享文件的同步互斥访问。

2、修改程序，观察对临界资源访问的互斥控制的作用。

1. **实验环境**

VMware15.5 PRO + Ubuntu 20.04

1. **实验内容及步骤(题目、相应程序代码、运行结果)**

1、（1）参照参考程序1，编写程序。父进程和两个子进程分别连续向共享文件中写入3行字符串。多次运行程序，观察共享文件内容。

#include<fcntl.h>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int fatal (const char\* info)

{

perror (info);

exit (1);

}

int lock(int fd)

{

lseek(fd,0,SEEK\_SET);

if(lockf(fd,F\_LOCK,0)==-1)

fatal("lockf()");

return 0;

}

int unlock(int fd)

{

lseek(fd,0,SEEK\_SET);

if(lockf(fd,F\_ULOCK,0)==-1)

fatal("unlockf()");

return 0;

}

int main()

{

int fd;

int p1,p2,i;

char str[20];

if((fd=open("locked\_file.txt",O\_RDWR|O\_APPEND|O\_CREAT,0666))<0)

fatal("open");

write(fd,"=========\n",10);

while((p1=fork( ))== -1); /\*创建子进程p1\*/

if (p1==0)

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str,"daughter %d\n",i);

write(fd,str,strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

}

else

{

while((p2=fork( ))==-1); /\*创建子进程p2\*/

if (p2==0)

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str,"son %d\n",i);

write(fd,str,strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

}

else

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str,"parent %d\n",i);

write(fd,str,strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

wait(NULL);

wait(NULL);

}

}

close(fd);

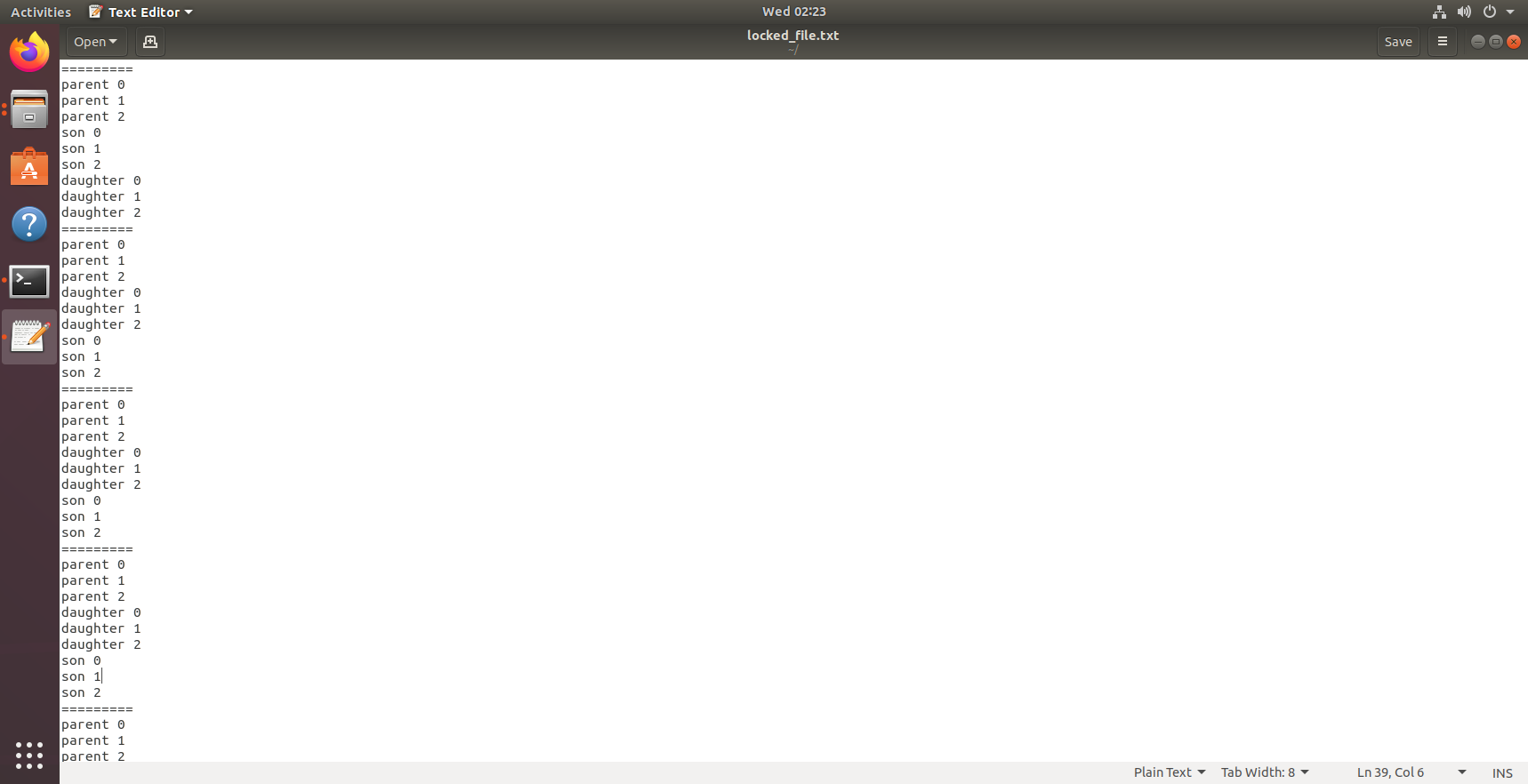
}

图1-1编写子程序1

（2）修改程序，去掉所有lock/unlock调用。多次运行程序，观察分析共享文件中，3个进程写入字符串的次序，解释原因

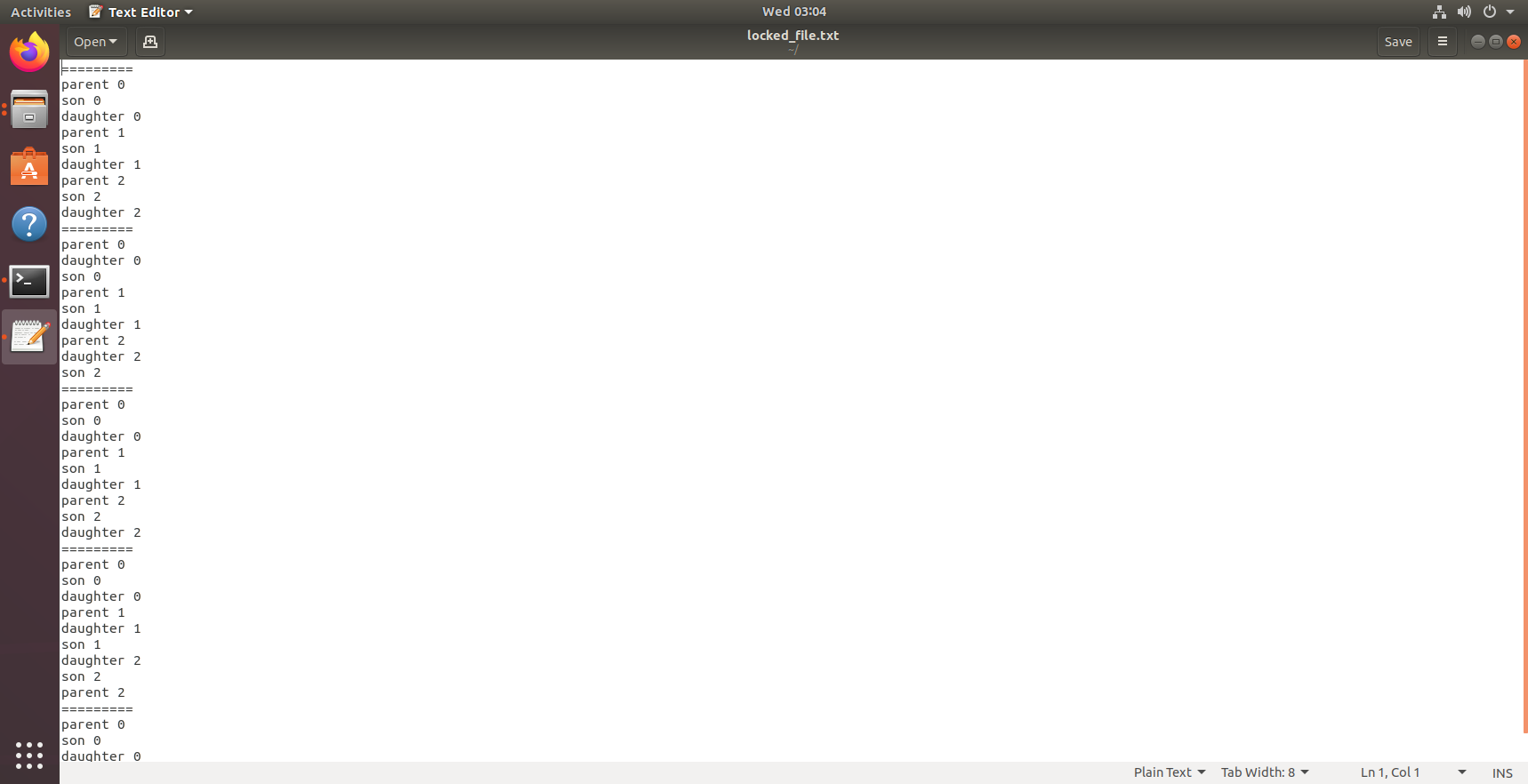


图1-2修改程序1，去掉lock/unlock

解释原因：从输出结果可以看出，去掉所有lock/unlock调用各个进程写入文件的顺序并不是按照加锁时间排序的，而是随机的。去掉程序中的lock和unlock调用，意味着多个进程可以同时访问共享文件，会导致数据错乱和不可预期的结果。因为不同进程间的写入是并发发生的，输出串的顺序无法确定。

2、（1）参照参考程序2，编写程序。父进程从外界获取字符串，并将其写入共享文件；子进程从共享文件中获取字符串，并将其打印出来。

#include<fcntl.h>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int fatal (const char\* info)

{

perror (info);

exit (1);

}

int lock(int fd)

{

lseek(fd,0,SEEK\_SET);

if(lockf(fd,F\_LOCK,0)==-1)

fatal("lockf()");

return 0;

}

int unlock(int fd)

{

lseek(fd,0,SEEK\_SET);

if(lockf(fd,F\_ULOCK,0)==-1)

fatal("unlockf()");

return 0;

}

int main( )

{

int pid,fd;

char str[80] = {0};

fd=open("tmp.txt",O\_RDWR|O\_CREAT|O\_TRUNC,0644);

pid=fork();

switch(pid)

{

case -1:

fatal("fork fail!");

case 0:

sleep(1);

lock(fd);

lseek(fd, SEEK\_SET,0);

read(fd,str,sizeof(str));

unlock(fd);

printf("son %d:read str from tmpfile:%s\n",getpid(),str);

exit(0);

default:

lock(fd);

printf("parent %d :please enter a str for tmpfile(strlen<80):\n",getpid());

//scanf("%s",str);

fgets(str, sizeof(str)-1, stdin);

lseek(fd, 0, SEEK\_SET);

write(fd,str,strlen(str));

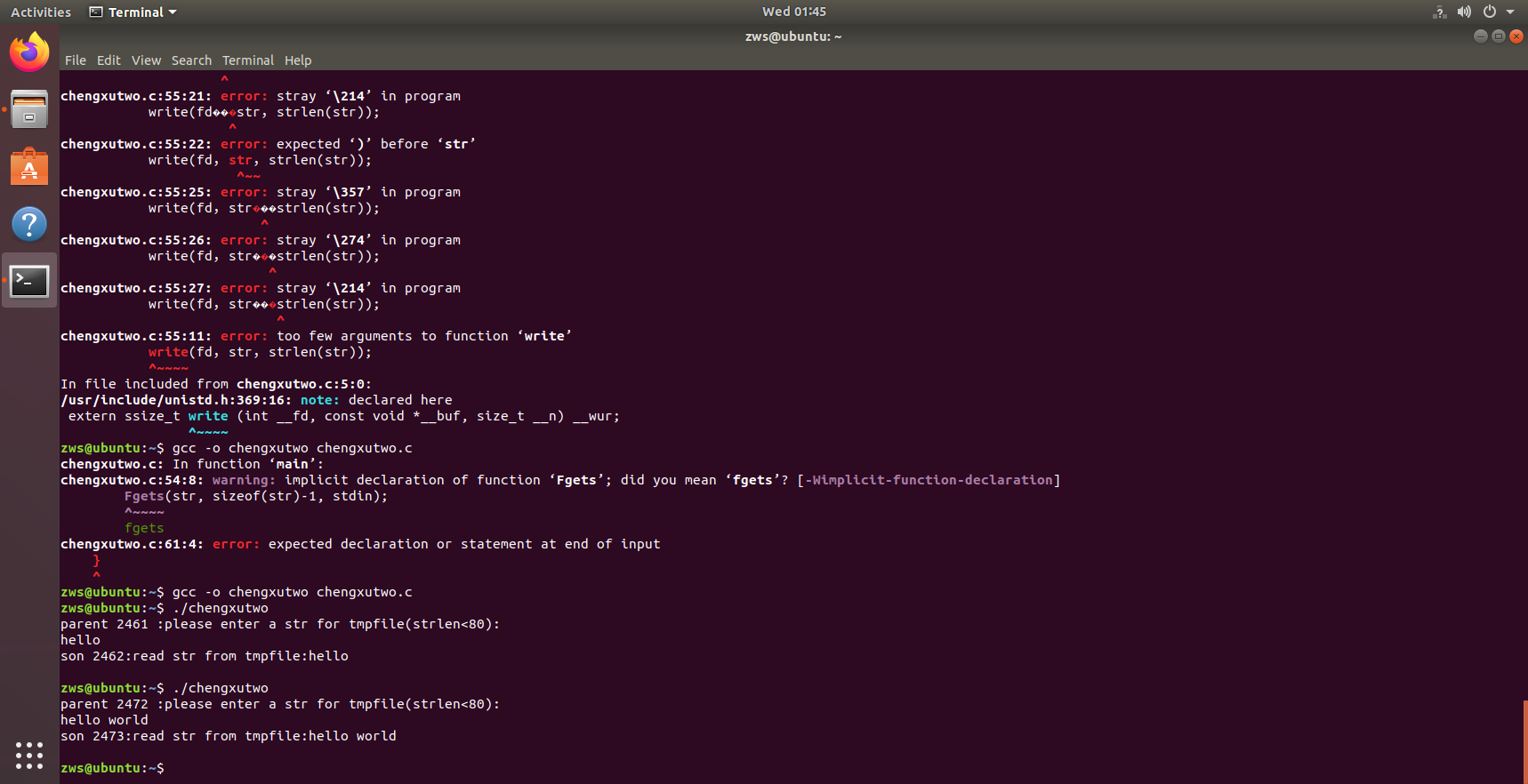
unlock(fd);

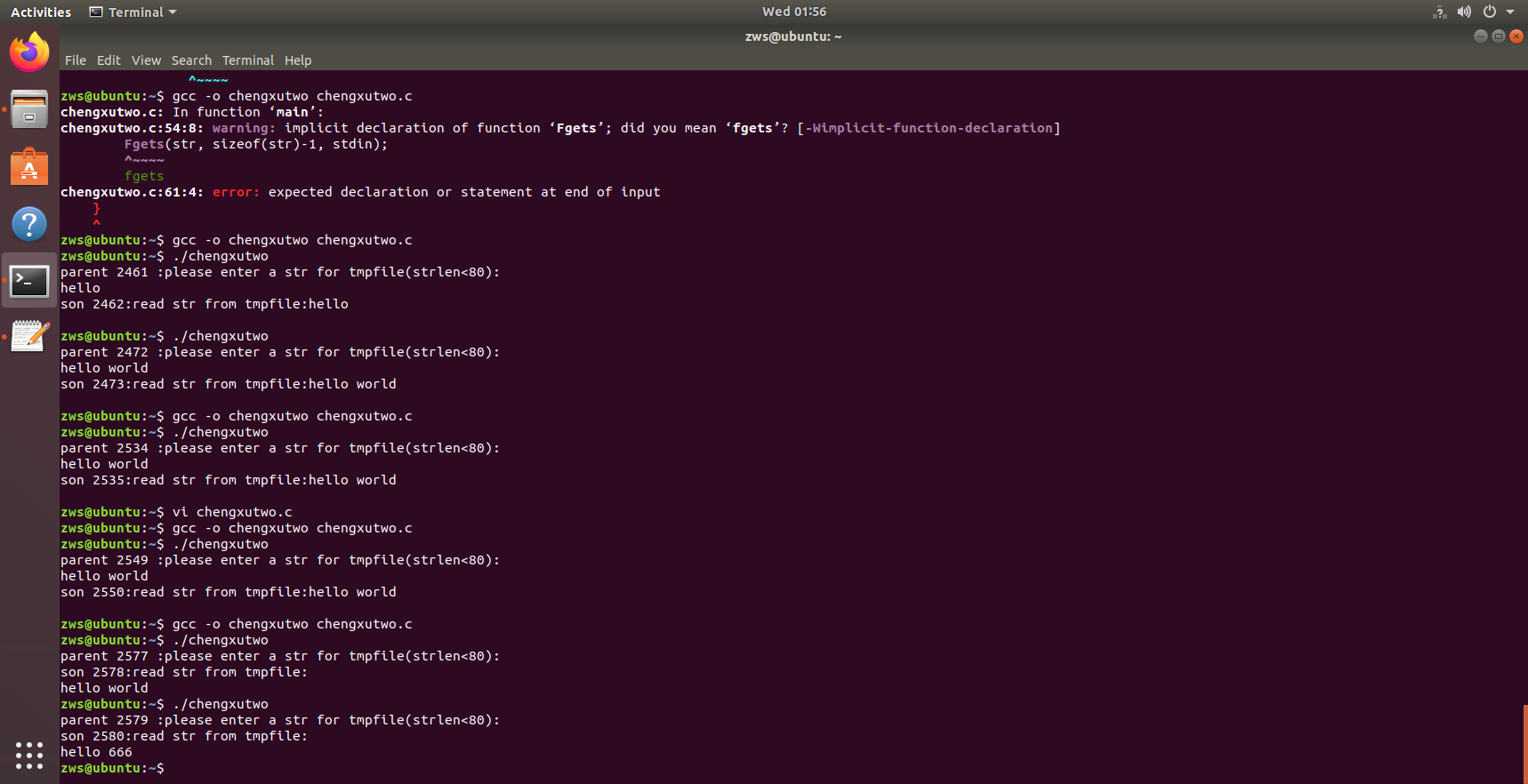
wait(0);

close(fd);

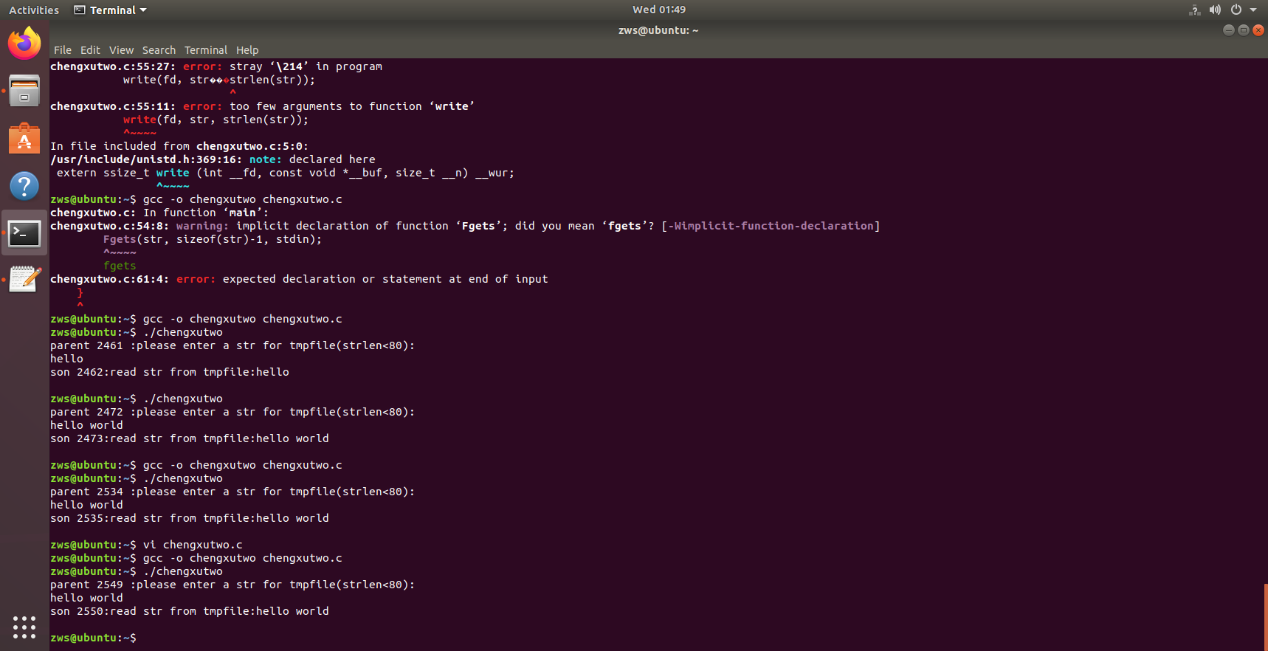
exit(0);

}

}

 （2）修改程序，去掉所有lock/unlock调用。观察分析运行结果，解释原因。

解释原因：读取的字符串为空，去掉所有lock()和unlock()调用后，运行程序，父进程正确地将字符串写入了共享文件，但子进程打印出的字符串为空，说明子进程读取共享文件时无法获取到正确的内容。这是因为共享文件的读写不受同步控制，父进程写入字符串的操作可能还没有完成，子进程就已经开始读取共享文件，此时读取的字符串为空。

 （3）修改程序，不是去掉lock/unlock调用，而是去掉子进程中的”sleep(1);”。观察分析运行结果，解释原因。

解释原因：去掉子进程中的sleep(1)后，运行程序，父子进程都可以正确地读写共享文件，说明在子进程读取共享文件之前，父进程已经完成了写入操作。因为fork()函数会复制父进程的所有资源，包括共享文件的描述符，子进程不会在打开共享文件之前创建一个新的文件，而是使用父进程打开文件的描述符。在父进程写入完成之前，文件指针已经被移动到末尾，因此子进程可以正确地读取共享文件中的内容。

1. **实验总结**

思考题

1. 函数unlock和lock的实现中，lseek的作用？

lseek()函数用于将文件读写指针移动到文件开头（SEEK\_SET）的位置，这是因为当读写指针位于文件中间或结尾时，调用lockf()函数可能会出现死锁的情况。所以在调用lockf()函数之前需要先将读写指针移动到文件开头，以确保加锁和解锁操作的正确性。当lseek调用成功时,返回值为一个字节为单位从文件头开始计算文件偏移量的值.调用失败时,返回值为-1. 文件指针又称文件读/写指针.文件刚打开时,文件指针指向开头位置;文件读写都是从文件指针处开始,并且在读写过程中同时移动文件指针.Lseek函数就是用于设置文件指针位置.

1. 解释参考程序1，2中lock/unlock的作用？

参考程序1和参考程序2中的lock()和unlock()函数是用于实现多进程之间对共享文件的读写同步控制的。当多个进程并发访问同一个共享文件时，需要对文件进行加锁和解锁操作，以保证在任一时刻只有一个进程可以对文件进行读写操作，防止数据出现混乱或数据丢失的情况。

1. 参考程序2，子进程中的”sleep(1);”的作用？

子进程中的sleep(1);语句是用于等待，以确保父进程先执行写入操作，从而保证在共享文件写入操作完成之后再读取操作。在不使用锁进行同步控制的情况下，为了避免父进程和子进程读写文件的时序问题，可以通过让父进程先执行写入操作再让子进程执行读取操作来解决此问题。

4、通过本次实验我进一步认识了并发执行的实质，学习到了解决进程同步互斥的方法，学会了使用相关函数来处理多个进程对共享资源的访问问题。加深了lockf(),sleep(),lock()等函数的具体运用。