目录

[实验2-1 2](#_Toc16901)

[1实验前准备：学习MAN命令的用法 2](#_Toc25591)

[2实验内容：编程实现软中断进程 3](#_Toc4564)

[1. 使用alarm进行中断： 3](#_Toc17489)

[2. 使用delete(ctrl+c)中断： 6](#_Toc21262)

[3. 讨论sleep(5)对结果的影响： 7](#_Toc25835)

[3实验报告预览： 7](#_Toc8971)

[1. 你最初认为运行结果会怎么样？写出你猜测的结果 7](#_Toc17330)

[4. 针对实验过程3，程序运行的结果是什么样子？时钟中断有什么不同？ 9](#_Toc28180)

[5.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？ 9](#_Toc12701)

[实验2-2 10](#_Toc10217)

[1实验前准备： 10](#_Toc10110)

[2 实验过程 10](#_Toc20932)

[1. 先猜想一下这个程序的运行结果。分析管道通信是怎样实现同步与互斥的 10](#_Toc9878)

[2. 然后按照注释里的要求把代码补充完整，运行程序； 10](#_Toc289)

[3. 修改程序并运行，体会互斥锁的作用，比较有锁和无锁程序的运行结果，并解释之。 10](#_Toc10920)

[3 实验报告预览 11](#_Toc30225)

[1. 你最初认为运行结果会怎么样？ 11](#_Toc8587)

[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。 12](#_Toc1256)

[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？ 12](#_Toc15520)

[4. 把README和程序源代码附到实验报告后 12](#_Toc30332)

[实验2-4 12](#_Toc21133)

[1 FIFO算法 12](#_Toc23386)

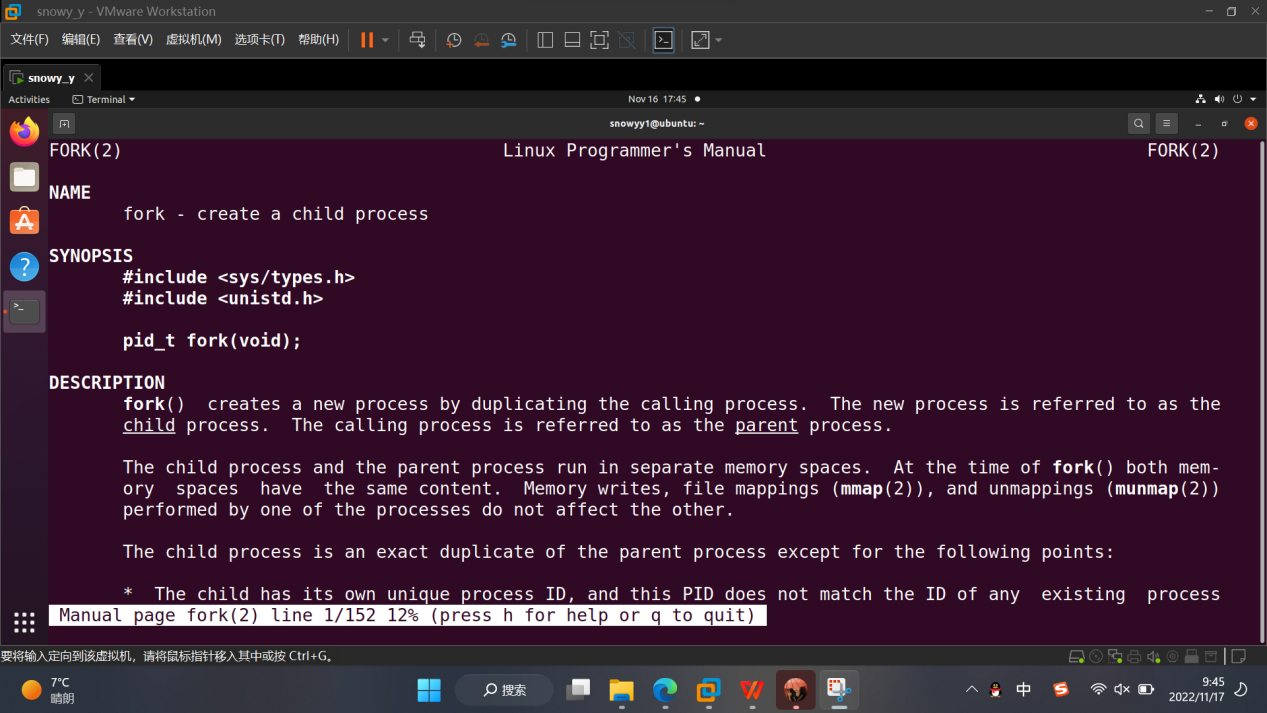
[2 LRU算法 13](#_Toc31783)

[3 源码 13](#_Toc29075)

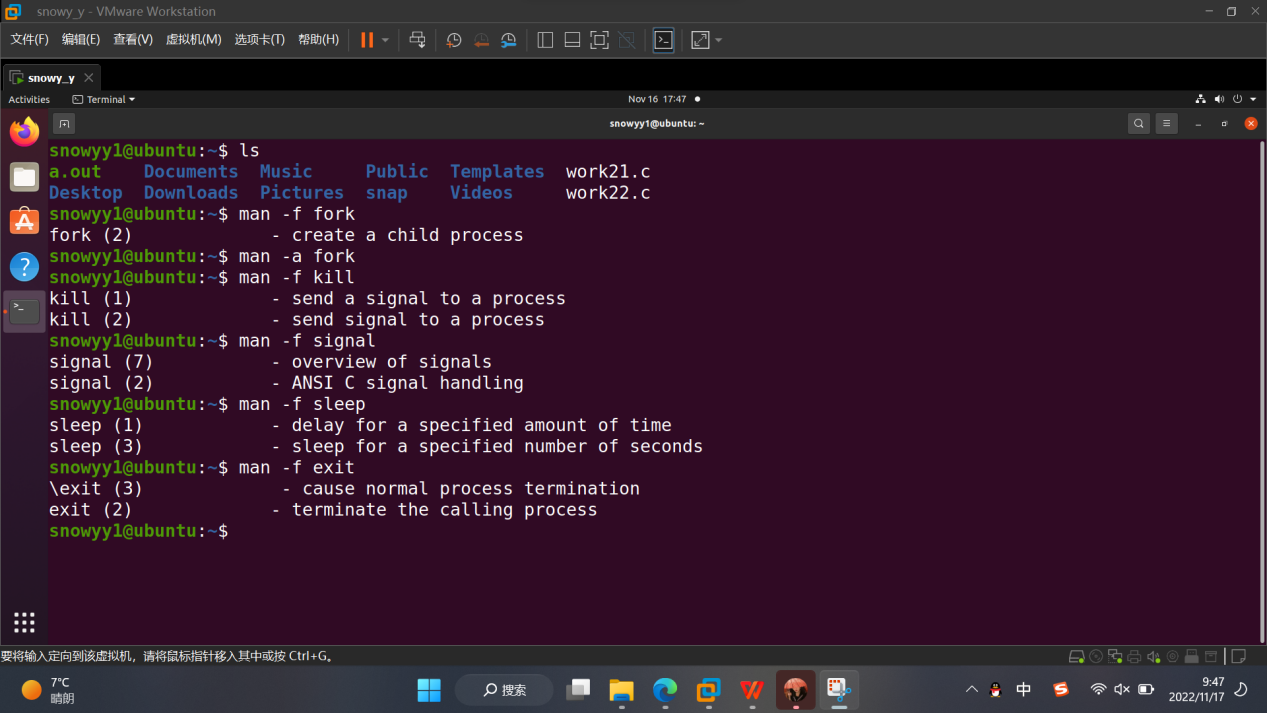
# 实验2-1

## 1实验前准备：学习MAN命令的用法

man -a fork



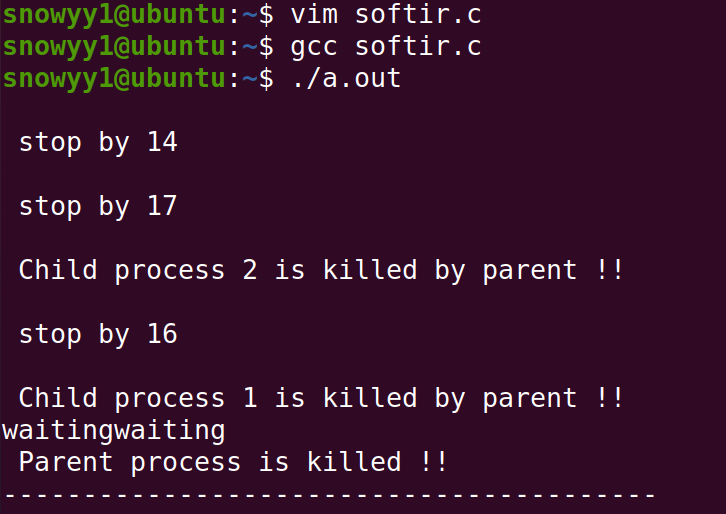
man -f 各种命令

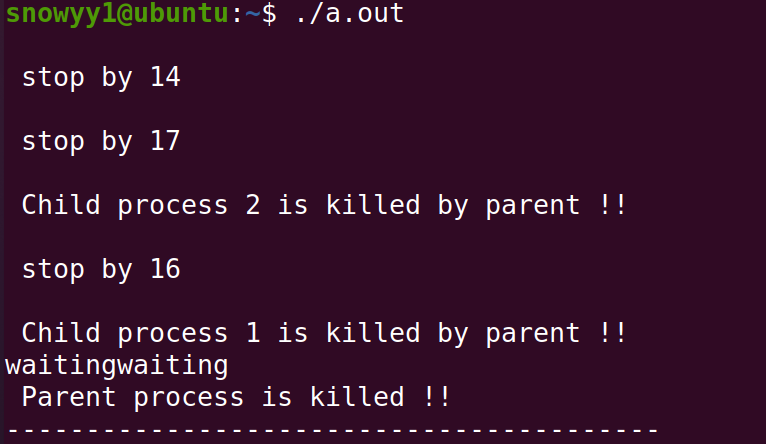


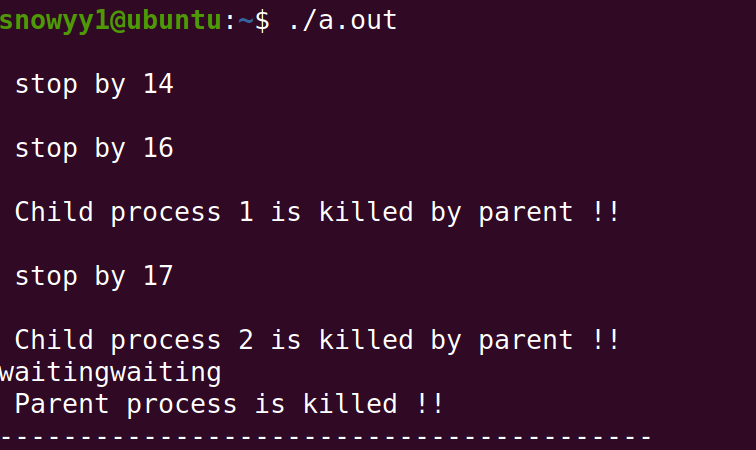
## 2实验内容：编程实现软中断进程

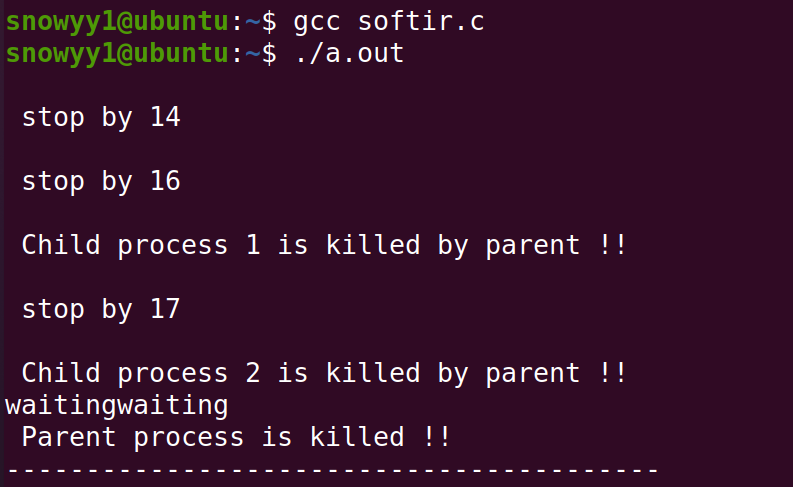
#### 使用alarm进行中断：

代码见附录文件softir.c





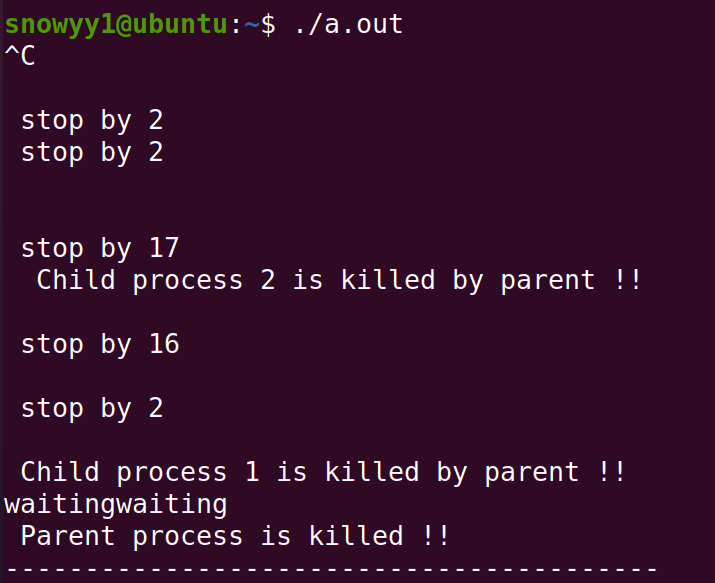
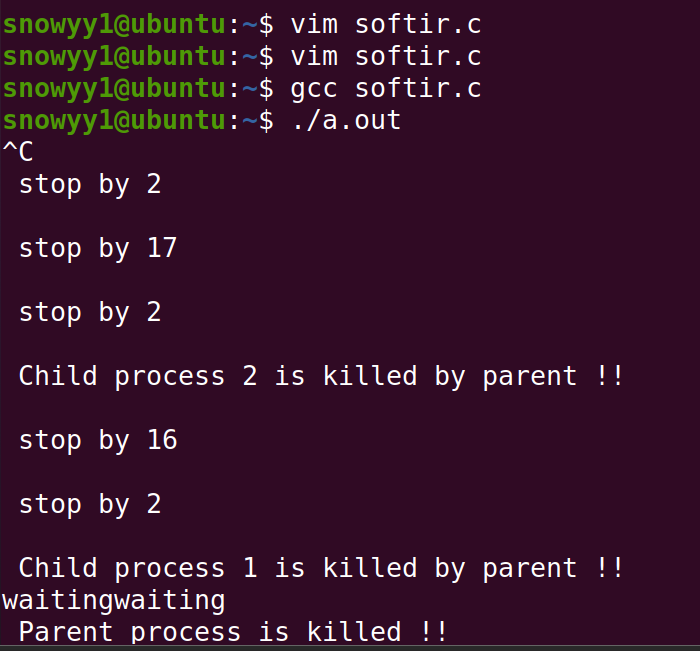




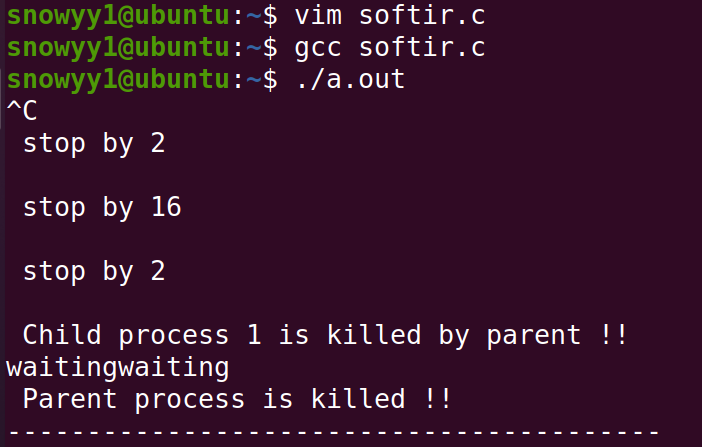
Ps:过程中出现的问题：无法解释’stop by’中间出现的换行

解决：通过在子进程“signal”语句前加入输出标识符，发现该输出标识符会代替回车，猜测该回车是自带机制。

#### 使用delete(ctrl+c)中断：



#### 讨论sleep(5)对结果的影响：



分析：三次参数为2的stop是因为signal(SIGINT,stop)；被继承了，三个进程都收到了信号。五次stop是因为有时结束得太快，在创建子进程之前就使得wait\_flag=1，这样进入子进程后继承的wait\_flag也是1，不用等到kill信号就自己停止了。同理也会出现执行四次的情况，与子进程创建的时机和按下delete的时机有关。

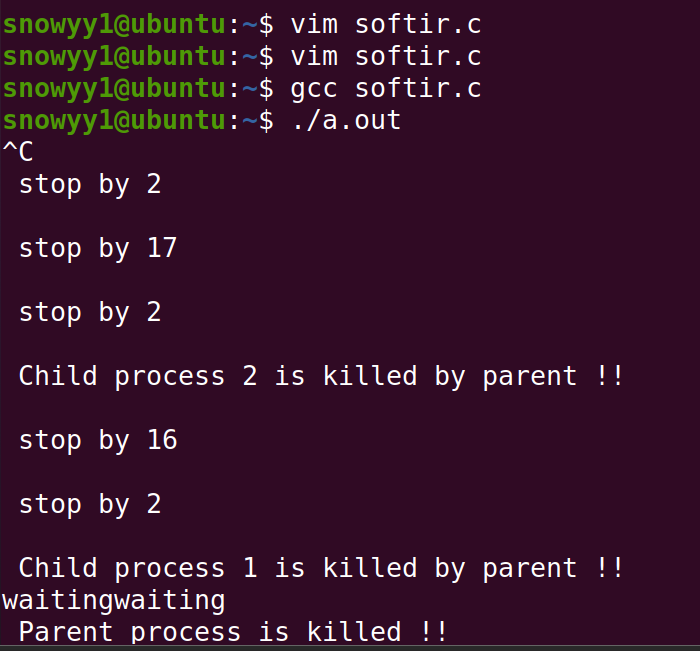
## 3实验报告预览：

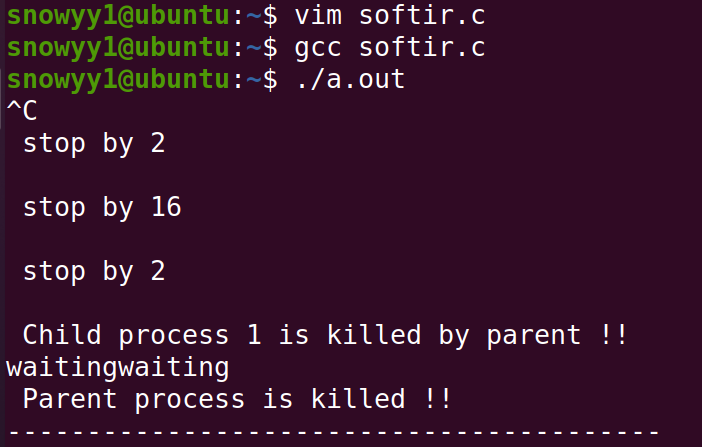
#### 你最初认为运行结果会怎么样？写出你猜测的结果

开始运行后应该开始无任何显示，等待键盘按下Delete键后显示“Child process 1 is killed by parent !! Child process 2 is killed by parent !!”，五秒之后显示“Parent process is killed !!”。

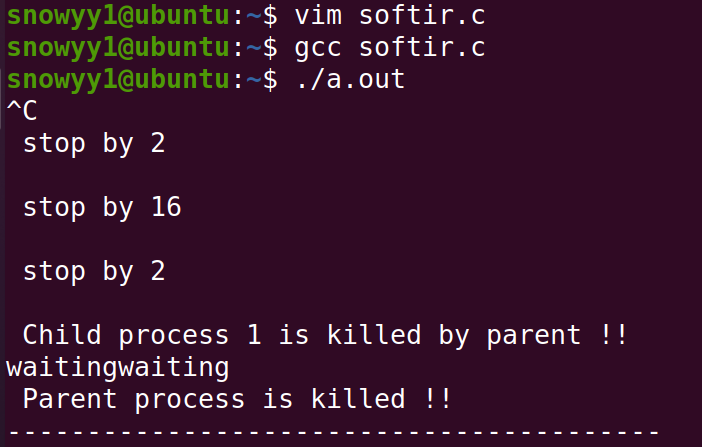
#### 实际的结果什么样？有什么特点？在接收不同中断前后有什么差别？请将5秒内中断和5秒后中断的运行结果截图，试对产生该现象的原因进行分析。

实际运行代码，发现与使用的中断方法无关，两个子进程结束的先后顺序不固定，因为两者的执行顺序由系统随机决定。同时，我在stop函数中添加了确认调用者的输出，进一步分析了整个过程的内部细节。

在5秒内中断时，结果如下：



而对于五秒后中断，结果如下：



#### 3 . 针对实验过程2，怎样修改的程序？修改前后程序的运行结果是什么？请截图说明。

截图见2，程序中通过加减sleep(5)操作实现结果变化

#### 4. 针对实验过程3，程序运行的结果是什么样子？时钟中断有什么不同？

分析：三次参数为2的stop是因为signal(SIGINT,stop)在子进程中也在被运行，一旦输入delete，三个进程会分别运行一次stop，三个进程都收到了信号。两次参数为2的stop是因为有时父进程结束得太快，子进程还没有运行复制来的signal(SIGINT,stop)就结束了。

#### 5.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？

kill命令使用了两次，分别给两个子进程以随机顺序发送16，17信号，但是子进程1先输出的概率大。通过将kill命令插入wait命令中间可以控制子进程执行顺序，因为父进程在第一个等待结束后才会发送下一个kill信号。（代码间softir.c有些地方被注释了）使用kill命令后，子进程接收到kill命令并调用stop函数，stop函数将wait\_flag置为0，输出被杀死的信号并结束。

#### 6.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？

进程调用return函数和exit函数可以主动退出。在两种退出方式中，主动退出比较好，如果在某个子进程退出前父进程被强制退出，则子进程会被init进程接管；如果用kill命令杀死某个子进程而其父进程没有调用wait函数等待，则该子进程为处于僵死状态占用资源。

# 实验2-2

## 1实验前准备：

利用博客了解write(),lockf()等函数

## 2 实验过程

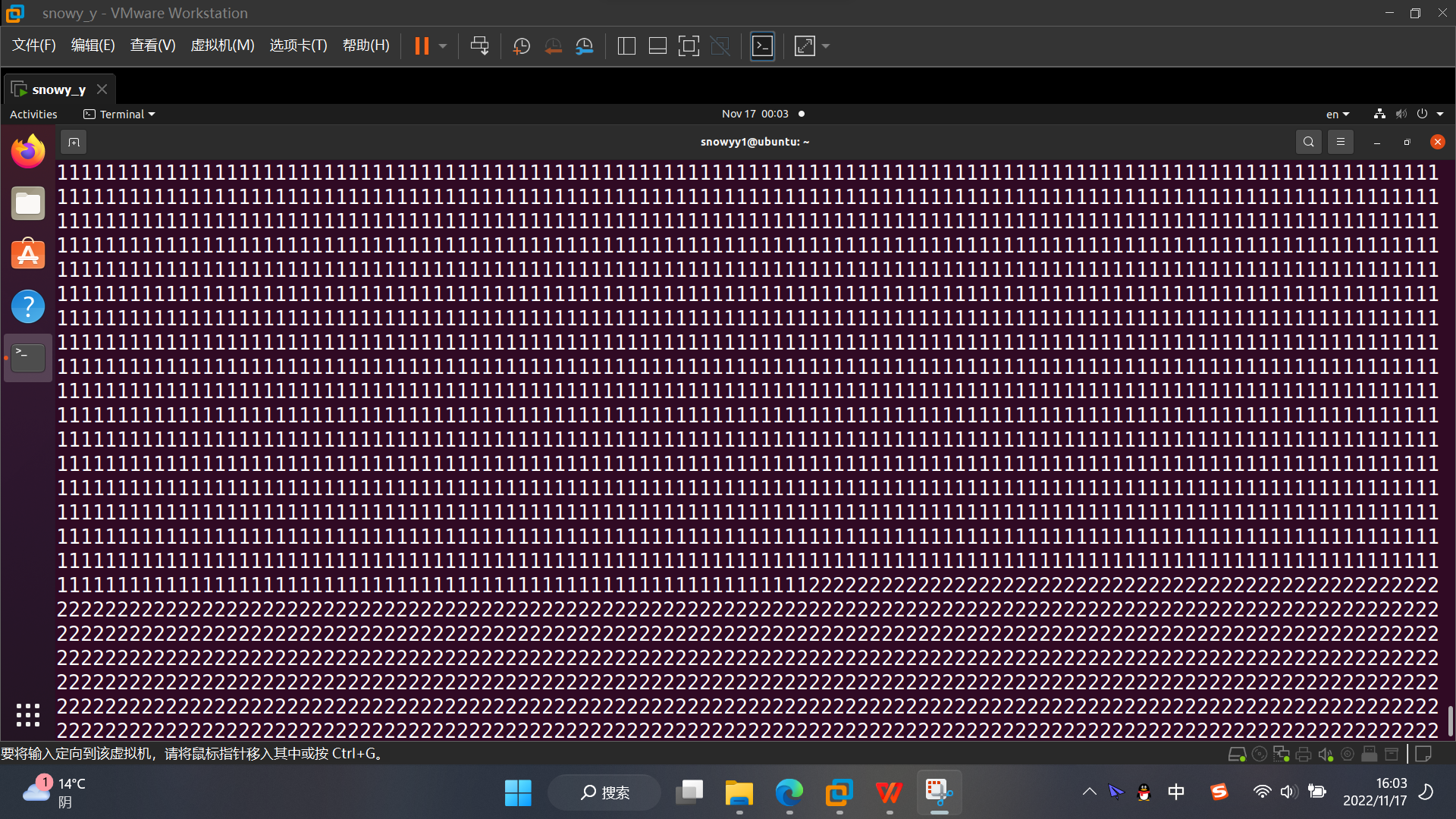
#### 先猜想一下这个程序的运行结果。分析管道通信是怎样实现同步与互斥的

终端会输出2000个1和2000个2但顺序不确定。

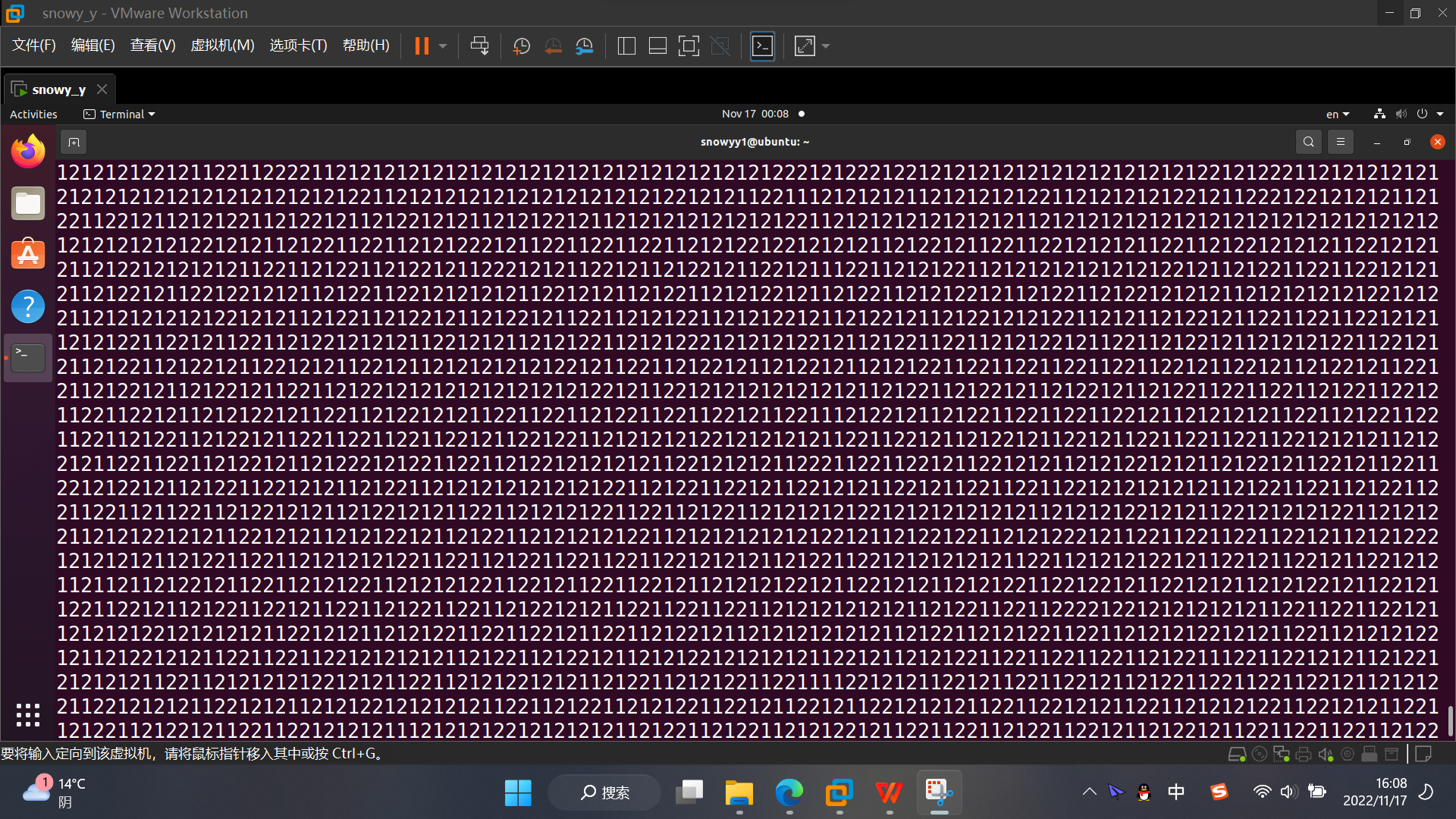
#### 然后按照注释里的要求把代码补充完整，运行程序；

#### 修改程序并运行，体会互斥锁的作用，比较有锁和无锁程序的运行结果，并解释之。

有锁运行结果如下：



去掉锁后结果如下：



Ps:刚开始并没有明显的12混合出现，在写入步骤后加上sleep(0.1)使结果更明显。

## 3 实验报告预览

#### 你最初认为运行结果会怎么样？

终端会输出2000个1和2000个2但顺序不确定。

#### 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。

输出了连续的2000个1和2000个2，两者先后不确定。

#### 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？

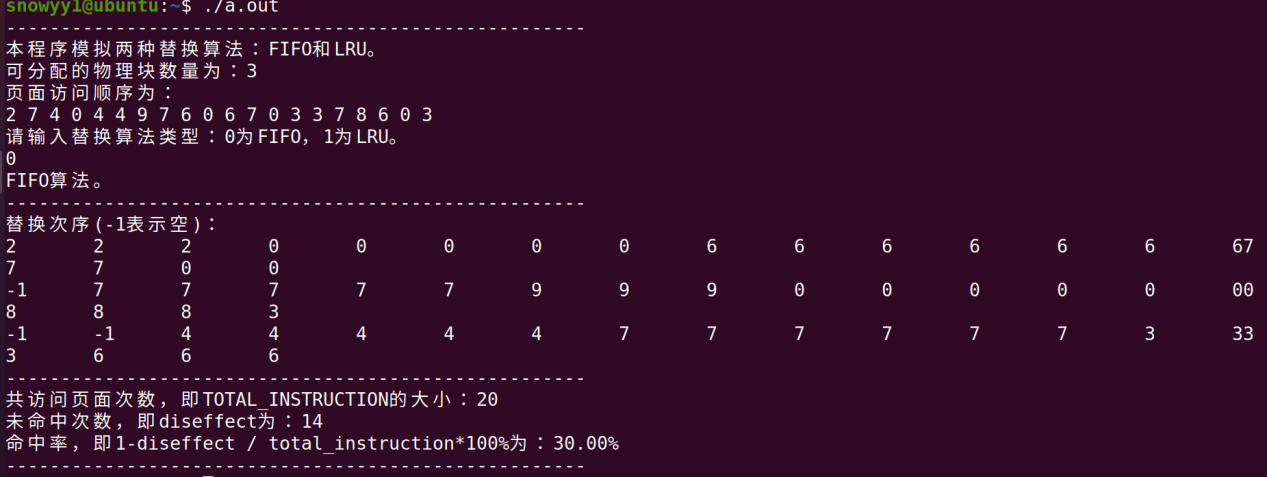
通过flock函数实现便捷的上锁和解锁，如果没有互斥操作，1、2会混乱的输出。

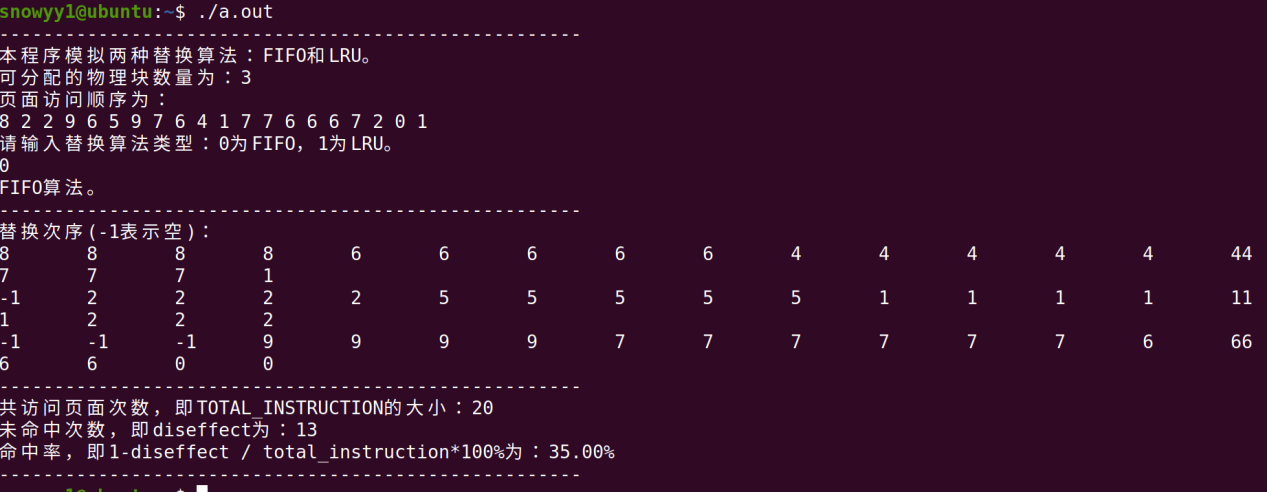
#### 把README和程序源代码附到实验报告后

源码见附件pipecom.c

# 实验2-4

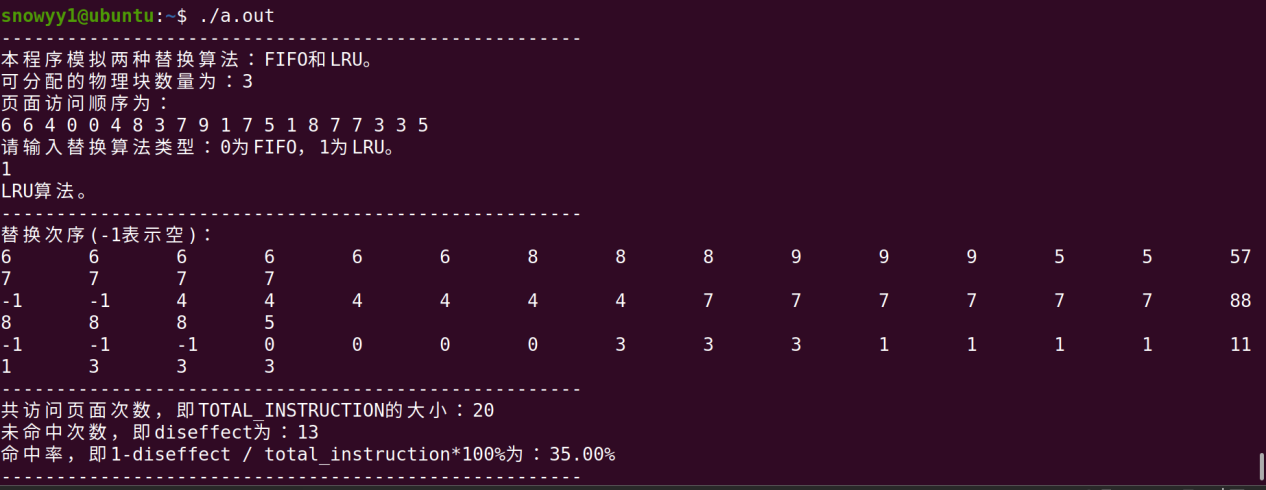
## 1 FIFO算法

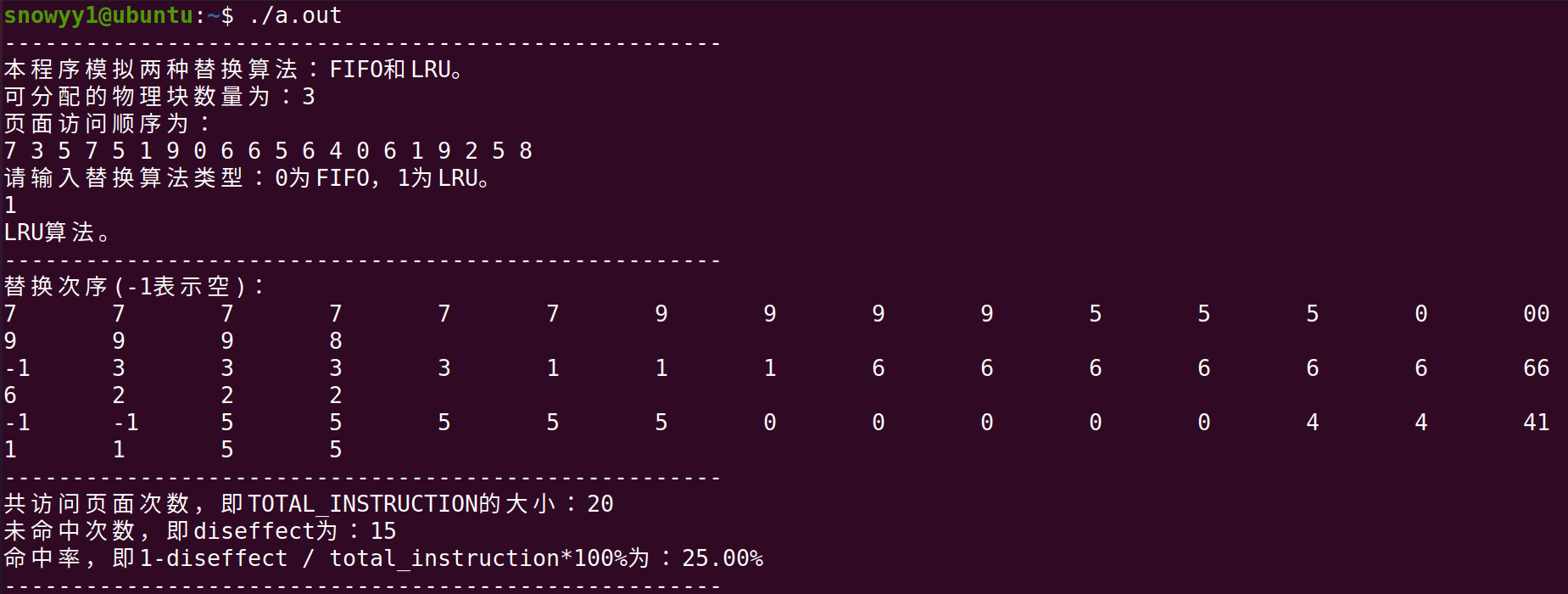




发现FIFO算法两次命中率分别为30%和35%

## 2 LRU算法





LRU算法两次命中率分别为35%和25%

## 3 源码

源码见附录文件pagechange.c