**操作系统原理**

**实验报告**

**学 号\_\_\_\_\_\_\_\_21071003\_\_**

**姓 名\_\_\_\_\_\_\_\_高立扬\_\_\_\_**

**提交日期\_\_\_\_\_2023.12\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **出勤情况** | **表现情况** | **考核情况** | **程序执行情况** | **问题回答情况** |
| **□按时出勤**  **□有缺席**  **□从未出现** | **□积极提问**  **□认真设计**  **□与同学一起商量**  **□主要靠别人讲解**  **□完全不参与** | **□演示实验代码**  **□进行答辩**  **□源代码分析** | **□顺畅**  **□有问题，经过老师指出之后改正**  **□有问题，无法改正** | **□立即正确回答**  **□经思考后正确回答**  **□回答有部分错误**  **□回答完全错误**  **□不能回答问题** |
| **报告结构** | **报告内容** | **报告图表** | **最终成绩** | |
| **□完全符合要求**  **□基本符合要求**  **□有比较多的缺陷**  **□完全不符合要求** | **□充实正确**  **□基本正确**  **□有一些问题**  **□问题很大** | **□符合规范**  **□基本符合规范**  **□有一定错误**  **□完全不正确** |  | |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

目录

[实验二 进程管理 3](#_Toc24098)

[一、 实验目的 3](#_Toc25679)

[二、 实验内容 3](#_Toc32554)

[三、 实验要求 4](#_Toc11235)

[四、 实验设计 4](#_Toc23952)

[4.1功能设计 4](#_Toc20977)

[4.2 数据结构 4](#_Toc1472)

[4.3 程序流程图 4](#_Toc28650)

[五、 实验结果及分析 5](#_Toc14171)

[六、 运行结果与感想 6](#_Toc19246)

[七、 参考资料 7](#_Toc7541)

[实验三 一个进程启动另一个程序的执行 7](#_Toc22734)

[一、 实验目的 7](#_Toc15844)

[二、 实验内容 7](#_Toc47)

[三、 实验要求 7](#_Toc32727)

[四、 实验设计 8](#_Toc7093)

[4.1功能设计 8](#_Toc27724)

[4.2 数据结构 8](#_Toc28647)

[4.3 程序流程图 8](#_Toc7972)

[五、 实验结果及分析 9](#_Toc21682)

[六、 运行结果与感想 10](#_Toc21988)

[七、 参考资料 10](#_Toc27054)

[实验四 基于消息队列和共享内存的进程间通信 10](#_Toc12640)

[一、 实验目的 10](#_Toc13710)

[二、 实验内容 10](#_Toc10725)

[三、 实验要求 11](#_Toc21035)

[四、 实验设计 11](#_Toc29092)

[4.1功能设计 11](#_Toc4552)

[4.2 数据结构 11](#_Toc8034)

[4.3 程序流程图 12](#_Toc12152)

[五、 实验结果及分析 12](#_Toc3001)

[六、 运行结果与感想 14](#_Toc27610)

[七、 参考资料 14](#_Toc23930)

[实验五 利用信号实现进程间通信 15](#_Toc16513)

[一、 实验目的 15](#_Toc26086)

[二、 实验内容 15](#_Toc26558)

[三、 实验要求 15](#_Toc13266)

[四、 实验设计 15](#_Toc9133)

[4.1功能设计 15](#_Toc15621)

[4.2 数据结构 16](#_Toc18097)

[4.3 程序流程图 16](#_Toc14147)

[五、 实验结果及分析 17](#_Toc20673)

[六、 运行结果与感想 17](#_Toc18140)

[七、 参考资料 18](#_Toc20535)

[实验六 线程的创建 18](#_Toc5839)

[一、 实验目的 18](#_Toc27137)

[二、 实验内容 18](#_Toc20139)

[三、 实验要求 19](#_Toc19551)

[四、 实验设计 19](#_Toc15578)

[4.1功能设计 19](#_Toc10392)

[4.2 数据结构 19](#_Toc2)

[4.3 程序流程图 19](#_Toc29512)

[五、 实验结果及分析 20](#_Toc4246)

[六、 运行结果与感想 21](#_Toc26681)

[七、 参考资料 21](#_Toc21461)

[实验七 利用信号量实现进程控制 22](#_Toc30180)

[一、 实验目的 22](#_Toc27584)

[二、 实验内容 22](#_Toc5639)

[三、 实验要求 22](#_Toc21458)

[四、 实验设计 22](#_Toc1020)

[4.1功能设计 22](#_Toc6282)

[4.2 数据结构 23](#_Toc1714)

[4.3 程序流程图 23](#_Toc10226)

[五、 实验结果及分析 23](#_Toc29556)

[六、 运行结果与感想 24](#_Toc27608)

[七、 参考资料 24](#_Toc30691)

# 实验二 进程管理

# 实验目的

加深对进程概念的理解，明确进程与程序的区别；进一步认识并发执行的实质。

# 实验内容

（1）进程创建

编写一段程序，使用系统调用fork()创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示“a“；子进程分别显示字符”b“和字符“c”。试观察记录屏幕上的显示结果，并分析原因。

（2）进程控制

修改已编写的程序，将每一个进程输出一个字符改为每一个进程输出一句话，再观察程序执行时屏幕上出现的现象，并分析原因。

（3）进程的管道通信

编写程序实现进程的管道通信。使用系统调用pipe()建立一个管道，二个子进程P1和P2分别向管道各写一句话：

Child 1 is sending a message!

Child 2 is sending a message!

父进程从管道中读出二个来自子进程的信息并显示（要求先接收P1，再接收P2）。

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

由所学可知：管道是进程间通信中最古老的方式，它包括无名管道和有名管道两种，前者用于父进程和子进程间的通信，后者用于运行于同一台机器上的任意两个进程间的通信。

1. 对于实验内容（1），首先创建2个pid\_t类型的变量来接收子程序的pid，接着就是调用fork()进行子程序的创建。规定父进程显示a，子进程分别显示b和c；
2. 将(1)中的进程输出内容更改即可
3. 对于实验内容（3），在（1）的基础上还要先进行pipe管道的创建，并且需要一个数组p[2]来控制管道的读写，p[0]控制读权限，而p[1]控制写权限。本实验仍然需要创建两个子程序，每一个子程序调用write()向管道中传输字符串。最后父进程调用read()来读取子进程1和2向管道中传输的字符串，因此还需要两个wait()来确保正确读取。

## 4.2 数据结构

本题目用到的管道pipe即为一种数据结构，它类似于一个队列，具有FIFO的性质，是单向数据流。

## 4.3 程序流程图

实验（1）（2）的程序流程图如下图1所示；实验（3）的程序流程图如下图2所示：

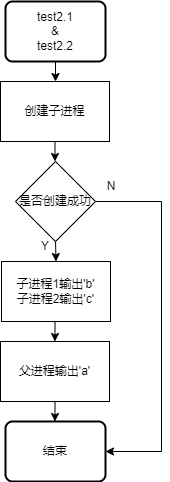
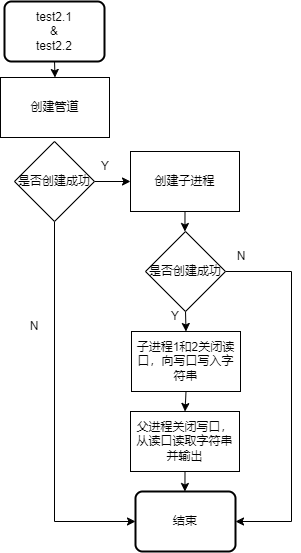
 

图1.test2.1&test2.2 图2.test2.3

# 实验结果及分析

对于实验(1)(2)，由于仅在输出内容上有改变，因此以下只针对(1)的结果开展说明。额外针对wait()的调用问题进行了研究：不添加wait()语句的程序运行结果如图3，在子程序2创建之前进行wait()的程序运行结果如图4：

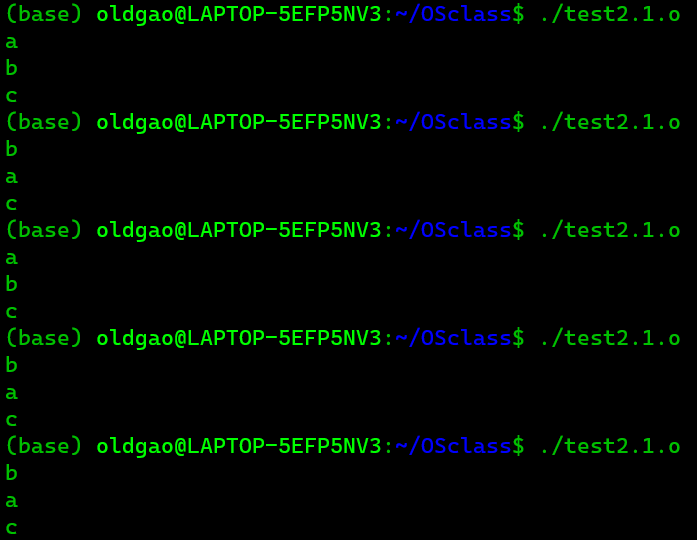
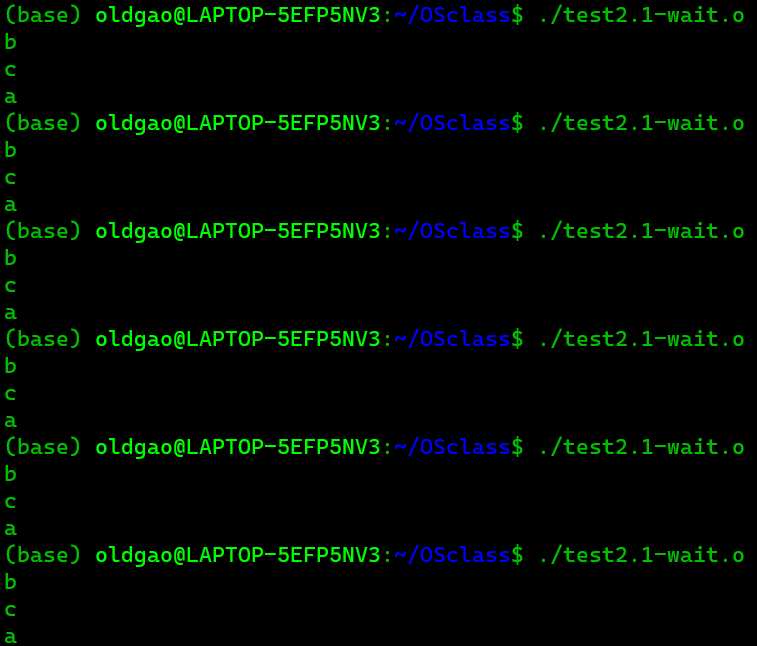
 

图3.test2.1 without wait() 图4.test2.1 with wait()

针对于wait()的研究，为实验内容(3)能正确地按照“父进程先接收P1,后接收P2”的要求。根据此研究，撰写实验内容(3)的代码，实验结果如下图5.：

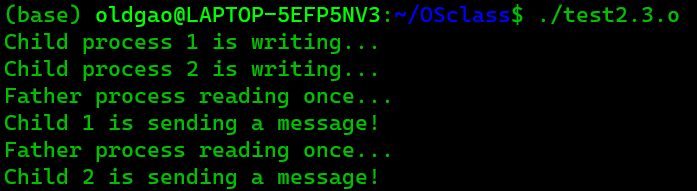


图5.test2.3

# 运行结果与感想

全部运行结果见下图6.：

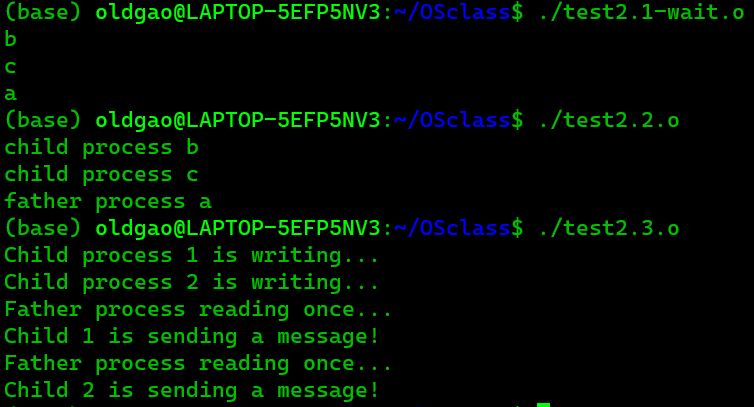


图6.实验二全部运行结果

本次实验，我首先针对wait()语句开展了研究：如果没有wait()，那么父子进程的输出将会是乱序的，这一实践加深了我对课本内容的强化记忆，并且使我明白了wait()的功能和调用时机，这为我之后的实验打下了基础，避免了很多问题的产生。

在实验内容（3），通过对管道pipe的学习，我学会了pipe读写操作，并且在控制读写权限数组时，体会到了“安全性”的重要性，如果一个pipe同时读写的话，那么将会导致非常多的问题产生。

# 参考资料

[1]OS实验指导书

# 实验三 一个进程启动另一个程序的执行

# 实验目的

编写Linux环境下，fork()与exec()的结合使用实现一个进程启动另一个程序的执行的基本方法，掌握exec()的几种调用方法。

# 实验内容

父进程从终端读取要执行的命令，并交给子进程执行。父进程等待子进程结束，并打印子进程的返回值。

提示：从终端读取要执行的命令可用fgets()实现。

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

设计一个死循环，在死循环中，程序不断接收用户输入的字符串，如果字符串为’exit’，那么就退出死循环并结束程序，否则就尝试执行字符串指向的程序或指令。调用exec()族中的execlp()，当输入字符串含’/’的时候，指令会尝试以目录的方式执行目标程序；反之，就会尝试执行系统指令。比如输入’./test.o’，那么程序就会尝试寻找根目录下叫test.o的文件并执行；输入’ls’，就会执行Linux指令ls。

## 4.2 数据结构

本实验无其余特殊数据结构

## 4.3 程序流程图

为了流程图简洁直观，突出针对实验要求的设计，以下实验流程图会忽略创建子进程是否成功的判断图。本实验程序流程图如下图7.所示：

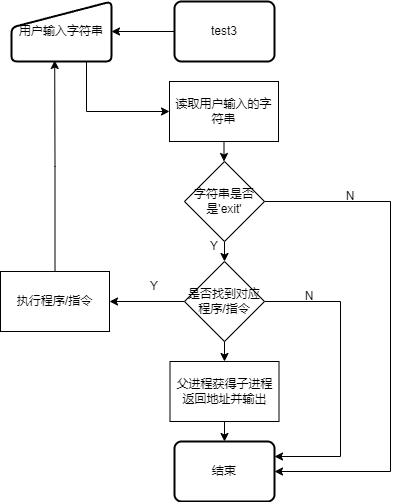


图7.test3流程图

# 实验结果及分析

如下图8.所示结果，首先执行ls指令，输出了目录下所有的文件，然后执行上一个实验的程序，输出了对应的结果，最后输入exit跳出循环终结程序:

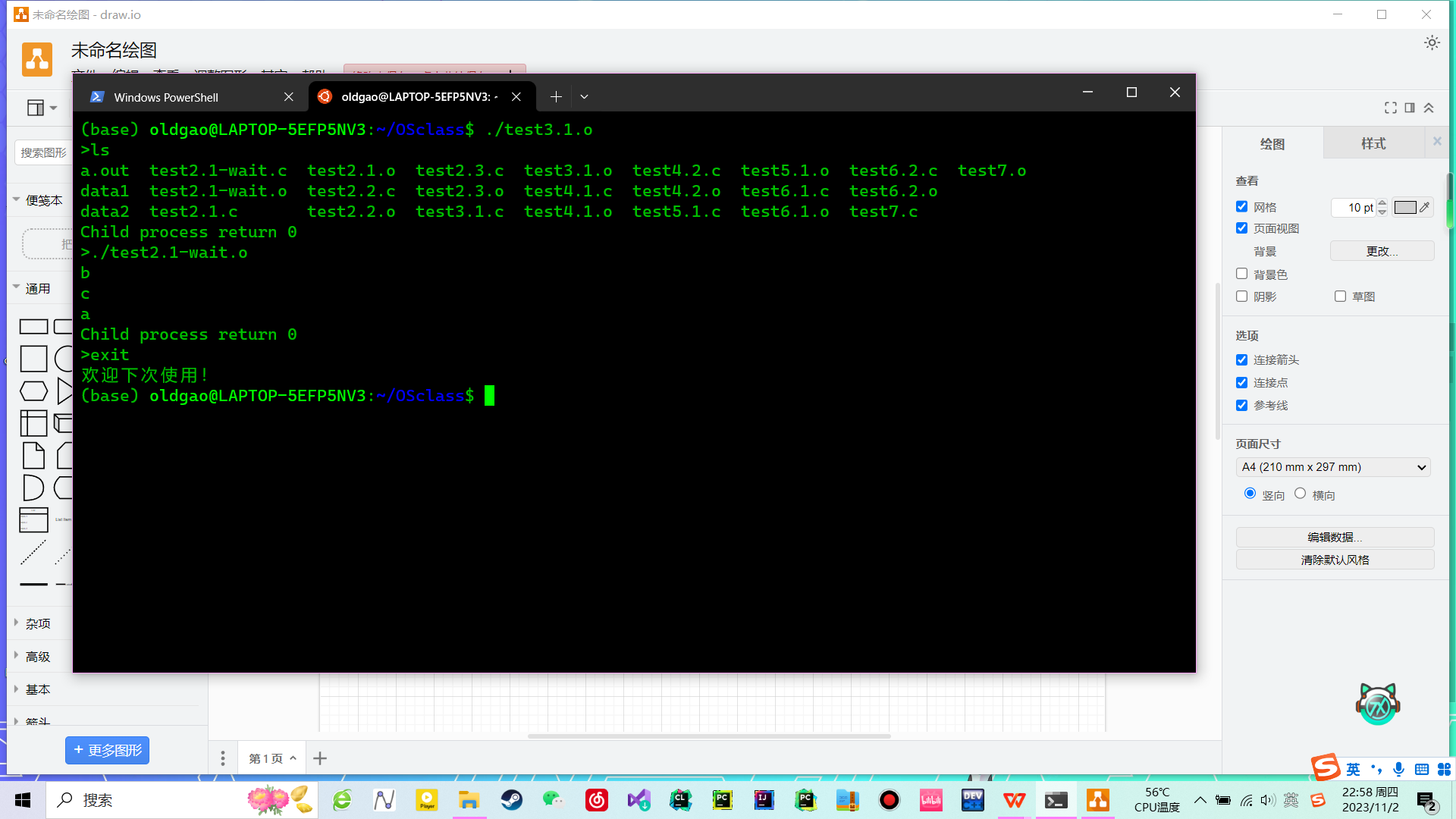


图8.test3实验结果

# 运行结果与感想

实验三的全部运行结果即为上图8.所示。

通过本次实验，我学习到了进程开启另一个进程的知识，并通过手动编写和debug实践完成了实验，并尝试分别调用了文件和系统环境里的指令。exec()族有很多函数，而我只接触了其中的一个，不过通过学习指导书内的函数说明以及上网查阅，发现这些函数功能近似，参数量不同，且参数基本上是围绕着寻文件地址和运行参数设置的。

# 参考资料

1. OS实验指导书
2. https://blog.csdn.net/m0\_70987863/article/details/128642948

# 实验四 基于消息队列和共享内存的进程间通信

# 实验目的

Linux系统的进程通信机构（IPC）允许在任意进程间大批量地交换数据。本实验的目的是了解和熟悉：

1. Linux支持的消息通信机制及其使用方法
2. Linux系统的共享存储区的原理及使用方法。

# 实验内容

* 1. 消息的创建、发送和接收

使用消息调用msgget（）、msgsnd（）、msggrev（）、msgctrl（）编制长度为1K的消息的发送和接收程序。

* 1. 共享存储取得创建、附接和断接

使用系统调用shmget（）、shmat（）、shmctl（）、shmctl（），编制一个与上述功能相同的程序。

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

为了便于操作和观察结果，用一个程序作为“引子”，先后fork()两个子进程，server和client，进行通信；

server端建立一个key为75的消息队列，等待其他进程发来的消息。当遇到类型为1的消息时，则作为结束信号，取消该队列，并退出Server。Server每收到一个消息后显示一句“（Server）received”；

Client端使用key为75的消息队列，先后发送类型为10到1的消息，然后退出。最后一个消息即server端需要的结束信号。Client每发送一条消息，显示一句“（Client）sent”；

父进程在server和client均退出后结束。

对于实验内容（1），首先要声明消息结构体和消息队列id，接着就是创建消息队列，然后是进行Client端和Server端，其中C端负责发送，而S端负责接收，当S端接收到消息类型为1的消息，立刻结束该队列并退出，最后程序运行完毕。创建消息队列的参数采用IPC\_CREATE。

对于实验内容（2），首先对需求进行分析：用共享存储区实现和实验内容（1）功能相同的程序，因此本实验重点在于共享存储区的应用，所以在代码层面可以省略消息结构体的声明，将重点落实在共享存储区的使用中。

值得注意的是：①共享存储区在创建之后还需要调用shmat()进行链接，用一个和共享存储区存储类型相同的指针②消息队列和共享存储区在程序结束前需要释放，分别调用msgctl()和shmctl()③创建消息队列或共享存储区的时候，除了填写本实验规定的key=75，还需要填权限参数，本实验填写0777，通过查阅资料了解到了0777的含义：

最左边的0代表了普通文件，目录d

左边的7代表了当前用户的权限，化为二进制数是111，分别代表了可读r可写w可执行x

中间的7代表了组用户的权限，定义同上

右边的7代表了其他用户的权限，定义同上

## 4.2 数据结构

消息结构体，其内部包括了type(int)代表消息类型、text(char\*)代表消息内容。规定type=1的消息是终止Server的消息。

## 4.3 程序流程图

实验内容（1）（2）的程序流程图分别见下图9-10.：

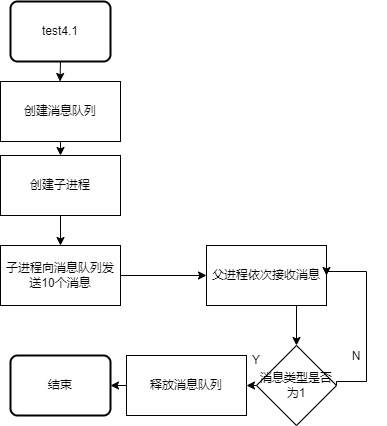
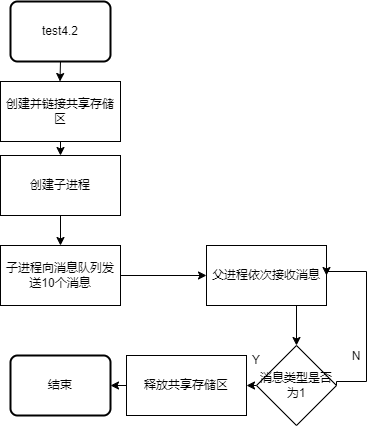
 

图9. test4.1 图10.test4.2

# 实验结果及分析

test4.1运行结果见下图11.所示。

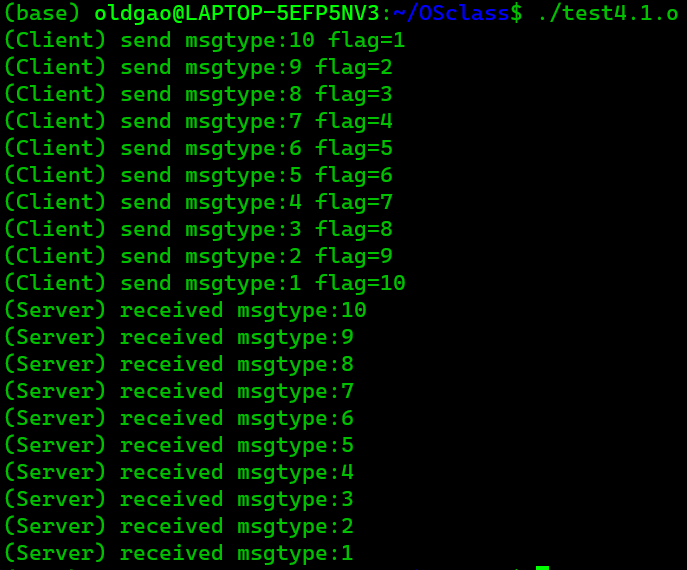


图11.test4.1运行结果

如本图可见，总共10个消息先被子进程统一传输进消息队列后，父进程才开始依次接收，并且是按照子进程传输顺序进行的接收，没有出现预想中子进程发送一次父进程接收一次的情况，因此对此做出实验：

①除去所有的wait()。但是实验结果图12.表明本次实验无效

②在①的基础上将msgsnd语句调换顺序。实验结果图13.表明在①的情况下msgsnd语句换到printf语句之前，会使得子进程发送和父进程读取交替运行（多次实验发现是乱序）

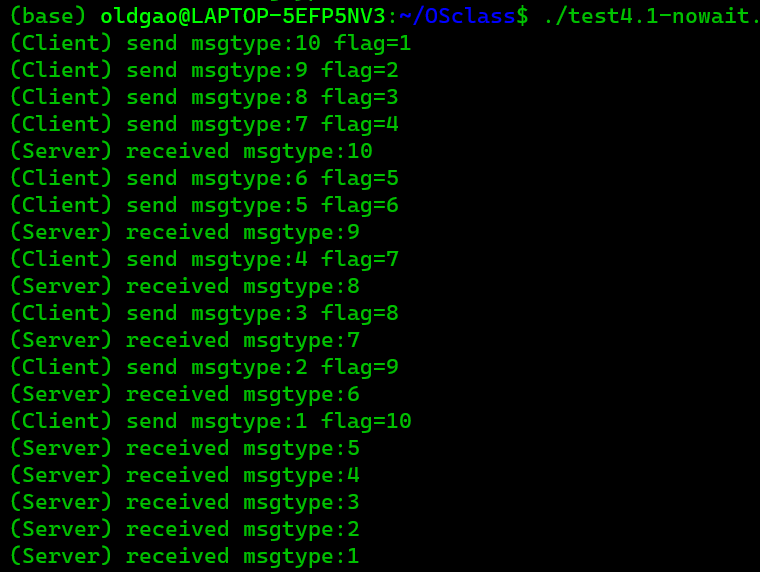
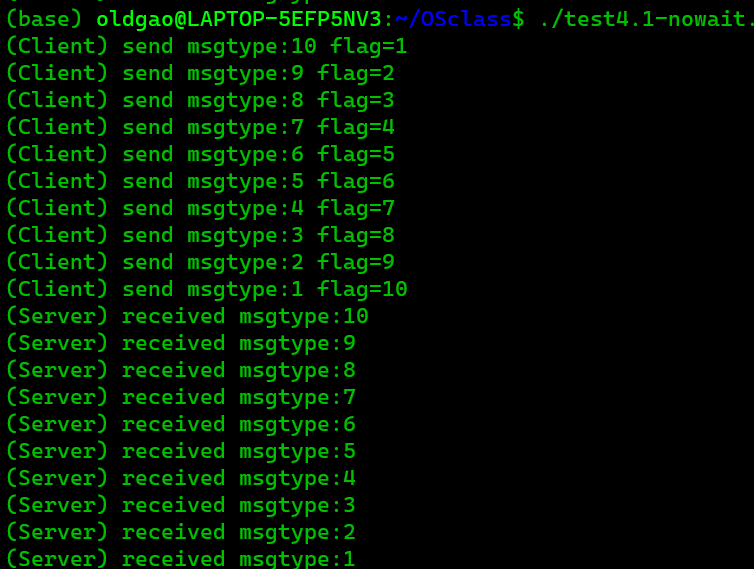


图12-13.实验①②运行结果

③test4.1原程序的msgsnd语句调换到printf语句之前，保留wait()。实验结果如图14.所示，表明本次实验无效

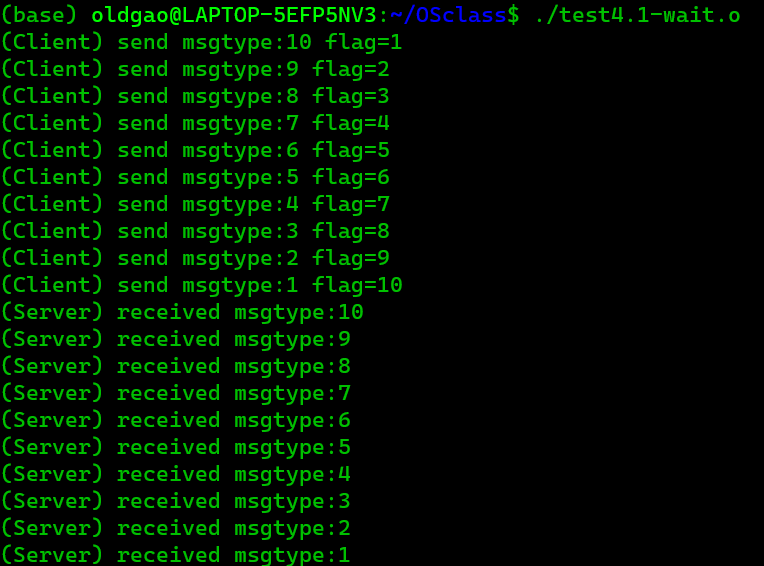


图14.实验③运行结果

总结上述实验发现，当wait()语句去除且msgsnd语句在printf之前的时候，才会得到预想中子进程发送与父进程接收交替运行的预期，奇怪的是只去除wait语句或是只调换msgsnd语句位置，从结果上来看都会出现原实验的结果，也就是父进程等待子进程消息全部发送完毕才会开始按顺序接收。

按照我所学的知识来看，printf仅仅是标准输出，并不会影响程序的运行结果，但是本实验确实又发生了printf改变程序运行结果的情况；test4.2没有在两个子进程创建之间写wait语句，却出现了乱序交替发送接收的情况（见六、中图16-17.），在撰写代码时我保证了test4.1和4.2除了必要代码不一致，其余代码及语句顺序都一致，因此实验结果令我更加疑惑，而查阅资料并没有查到相关话题，因此只能保留这个疑问，留给后人进行研究。

# 运行结果与感想

代码中子进程按顺序发送了type为10到1的消息，因此父进程接收时忽略了type=1的判断。下面是实验的全部运行结果如图15-17：

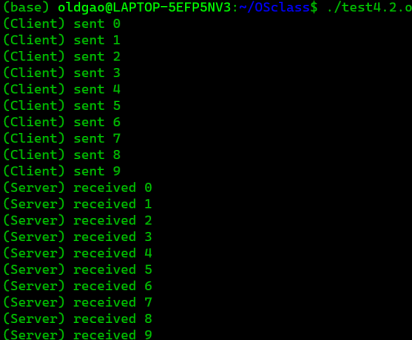
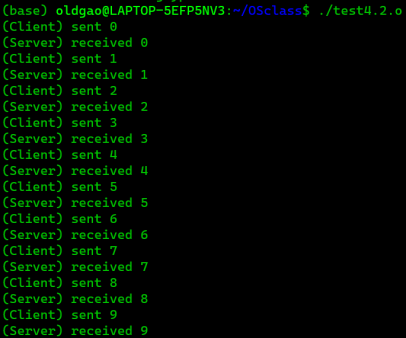
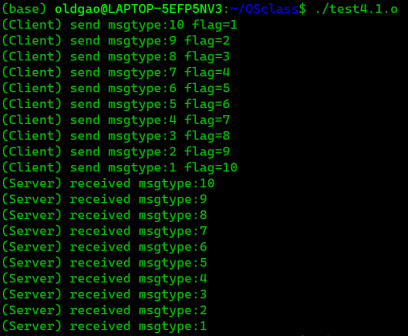


图15-17.实验四全部实验结果

本次实验我通过思考和实践，发现了消息队列msgsnd语句和printf语句顺序会影响输出结果的问题，虽然没有查阅到相关资料，但是实验结果却表明了上述问题并非虚构。鉴于以上实践结果，我更推荐利用共享存储区来进行消息传递。

# 参考资料

1. OS实验指导书
2. https://wenku.baidu.com/view/80c838880329bd64783e0912a216147917117e15.html?\_wkts\_=1698977446744&bdQuery=msgget+0777%E7%9A%84%E5%90%AB%E4%B9%89

# 实验五 利用信号实现进程间通信

# 实验目的

学习UNIX类操作系统信号机制，编写Linux环境下利用信号实现进程间通信的方法，掌握注册信号处理程序及signal()调用方法。

# 实验内容

编写一个程序，完成下列功能：实现一个SIGINT信号的处理程序，注册该信号处理程序，创建一个子进程，父子进程都进入等待。SIGINT信号的处理程序完成的任务包括打印接受到的信号的编号和进程PID。编译并运行该程序，然后在键盘上敲Ctrl + C，观察出现的现象，并解释。

提示：参见“五、补充材料”中的signal()的基本用法。

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

信号（signal）是UNIX提供的进程间通信与同步机制之一。信号用于通知进程发生了某个异步事件。信号与硬件中断相似，但不使用优先级。即，认为所有信号是平等的；同一时刻发生的多个信号，每次向进程提供一个，不会进行特别排序。

进程可以相互发送信号，内核也可以发出信号。信号的发送通过更新信号接收进程的进程表的特定域而实现。由于每个信号作为一个二进制位代表，同一类型的信号不能排队等待处理。

信号处理函数sig\_user()参考了实验指导书中的源代码，而main函数则只创建了USR1子程序便于核心功能实践。子程序调用signal函数，第一个参数是SIGUSR1，父进程调用signal函数，第一个参数是SIGINT，等待用户输入Ctrl+C并终止程序，否则一直等待。在signal调用结束后，用for(::) pause()进行等待。

## 4.2 数据结构

本实验无其余特殊数据结构

## 4.3 程序流程图

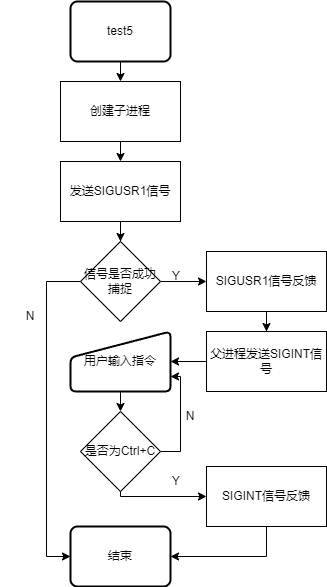


图18.test5.1

# 实验结果及分析

实验结果如图19.所示。由图19.可以看到，子进程USR1被创建，父进程被创建，此时进入用户Ctrl+C指令的输入等待，直到正确输入才会退出。此时用ps查看进程状态，发现父子进程全部结束，符合预期（如图20.）

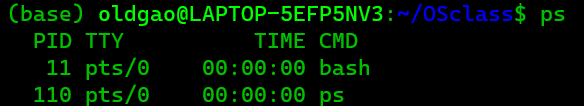
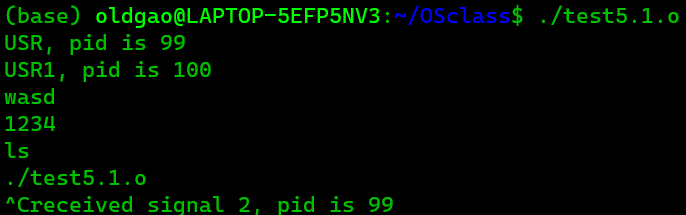


图19.test5实验结果 图20.程序全部结束

下面尝试后台运行，实验结果如图21.，用指导书中的指令kill -USR1 pid终止子进程，发现信号成功被反馈，但是由于父进程还没有结束，等待仍在进行，父子进程都未终止（如图21.）。至此实验结束

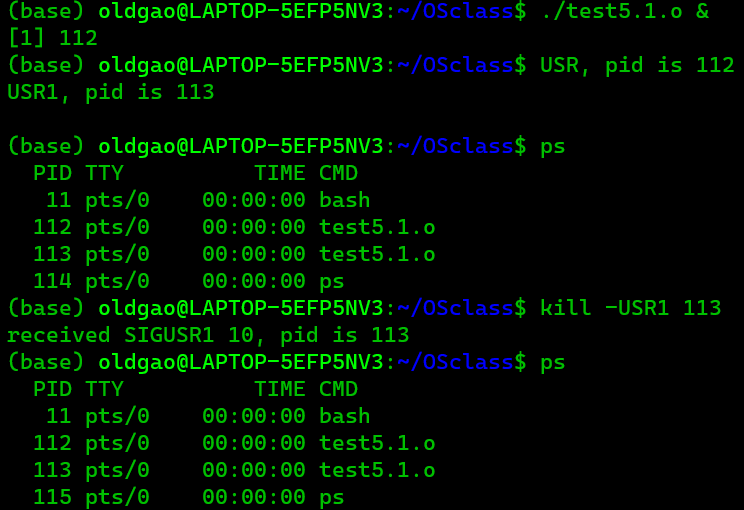


图21.后台运行实验结果

# 运行结果与感想

实验五的运行结果如图22.所示：

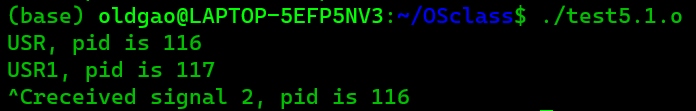


图22.test5运行结果

本次实验我接触到了后台运行，学习并实践了利用信号在进程之间通信，并且了解到了多种功能各异信号。我认为信号通信十分便捷，而且不用手动释放，为进程通信带来了便利。

# 参考资料

1. OS实验指导书
2. UNIX/Linux操作系统的联机帮助（使用man命令）：signal(2)

# 实验六 线程的创建

# 实验目的

编写Linux环境下的多线程程序，了解多线程的程序设计方法，掌握最常用的三个函数pthread\_create，pthread\_join和pthread\_exit的用法

# 实验内容

1、主程序创建两个线程myThread1和myThread2，每个线程打印一句话。使用pthread\_create(&id,NULL,(void \*) thread,NULL)完成。

提示：

先定义每个线程的执行体，然后在main中()创建几个线程，最后主线程等待子线程结束后再退出。

2、创建两个线程，分别向线程传递如下两种类型的参数

·传递整型值

·传递字符

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

　Linux下的多线程遵循POSIX线程接口，称为pthread。编写Linux下的多线程程序，需要使用头文件pthread.h，连接时需要使用库pthread。Linux下pthread的实现是通过系统调用clone（）来实现的。clone（）是Linux所特有的系统调用，它的使用方式类似fork，关于clone（）的详细情况，可有关文档说明。

注意：编译时要使用如下命令（设example.c是源程序名字）。因为pthread的库不是linux系统的库，所以在进行编译的时候要加上-lpthread，否则编译不过。具体命令如下：

gcc example.c -lpthread -o example

对于实验内容（1），首先声明两个自定义线程函数（无参），类型为void（因此在创建进程时需要类型转换），接着写main函数，在main函数内先声明pthread\_t类型的id1和id2用于接收线程id，并声明int型的ret变量接收返回值。然后依次创建两个线程，当线程返回值非0则报错。最后，由于父线程需要等待子线程，所以需要分别在线程1和2创建后调用pthread\_join()。

对于实验内容（2），首先声明两个自定义线程函数（参数为\*arg），类型为void\*（无需类型转换），接着在main函数中，在（1）的基础上添加int\*和char\*指针用于参数传递，其余效仿（1）撰写程序代码即可。

## 4.2 数据结构

本实验无其余特殊数据结构

## 4.3 程序流程图

实验内容（1）（2）的流程图均可以用下图23.统一表示：

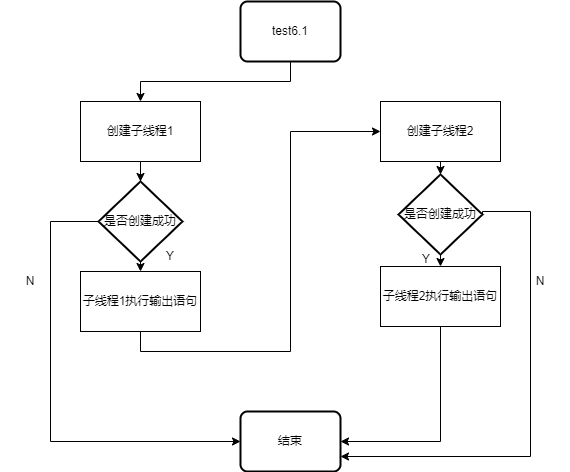


图23.test6流程图

# 实验结果及分析

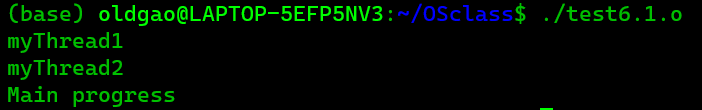
实验以实验内容（1）为示例在下文展示。我主要针对pthread\_join进行了实验测试：

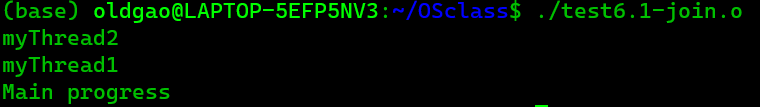
①pthread\_join等待p1的语句写在p2创建之前

②pthread\_join等待p1的语句写在p2创建之后

③无pthread\_join

实验结果如下图24-26.所示：





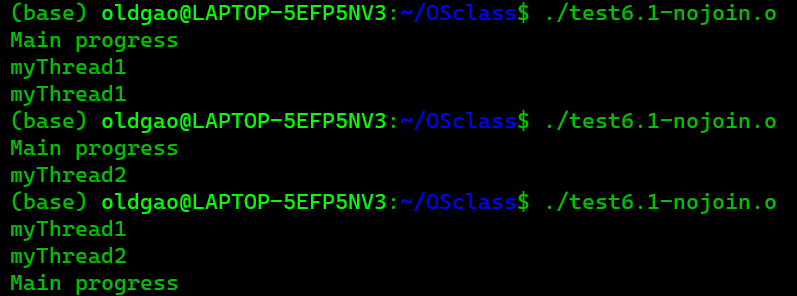
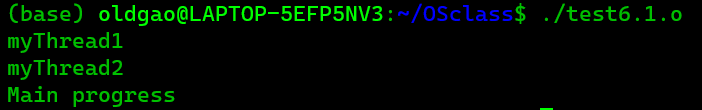


图24-26.三个实验的结果

由第一张图是符合该实验要求的标准结果。第二张图对应②，有一定概率线程2先于线程1结束，这说明若严格规定每一个线程的先后顺序，必须要在每下一个线程创建之前进行pthread\_join()。第三张图对应③，可以发现如果不写pthread\_join()会导致线程执行顺序混乱，甚至出现进程创建但不执行（因为创建失败会有对应输出，上图没有），进程重复创建执行的情况。通过man pthread\_join可以得知，pthread\_join是负责线程安全性的函数，如果一个线程没有join，那么就无法保证线程的正确执行和终止。

# 运行结果与感想



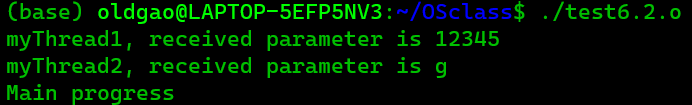


图27-28.test6运行结果

本次实验我通过对pthread\_join的探究，发现了其在多线程编程中的重要性与必要性，并灵活运用man指令，脱离了对网络资料的依赖。

# 参考资料

1. OS实验指导书
2. UNIX/Linux操作系统的联机帮助（使用man命令）：pthread\_join

# 实验七 利用信号量实现进程控制

# 实验目的

学习UNIX类操作系统信号量机制，编写Linux环境下利用信号量实现进程控制的方法，掌握相关系统调用的使用方法。

# 实验内容

创建4个线程，其中两个线程负责从文件读取数据到公共的缓冲区，另两个线程从缓冲区读取数据作不同的处理（加和乘运算）。使用信号量控制这些线程的执行。

提示：参见“五、补充材料”中的相关系统调用的基本用法。

# 实验要求

按照要求编写程序，放在相应的目录中，编译成功后执行，并按照要求分析执行结果，并写出实验报告。

# 实验设计

## 4.1功能设计

信号量本质上是一个非负的整数计数器，它被用来控制对公共资源的访问。当公共资源增加时，调用函数sem\_post（）增加信号量。只有当信号量值大于０时，才能使用公共资源，使用后，函数sem\_wait（）减少信号量。函数sem\_trywait（）和函数pthread\_ mutex\_trylock（）起同样的作用，它是函数sem\_wait（）的非阻塞版本。

本次实验我主要用到的是sem\_wait()和sem\_post()，它们便是课程中讲到的P和V，因此先用define重命名它们，会让代码看起来更加直观好理解。在开始写函数之前，先要进行必要变量的准备：MAXLEN=10，buffer[MAXLEN]数组用来模拟缓存区，push\_flag = -1用来模拟压栈，还有sem\_t类型的变量b,mutex,r，其中b代表了缓存区剩余资源量，初始数设置为MAXLEN，mutex负责互斥访问，初始数值设置为1，r确保线程不读取空缓存区，因此初值为0。data1文件存储“100 100”，data2文件存储“60”

## 4.2 数据结构

buffer数组，用来模拟缓冲区

## 4.3 程序流程图

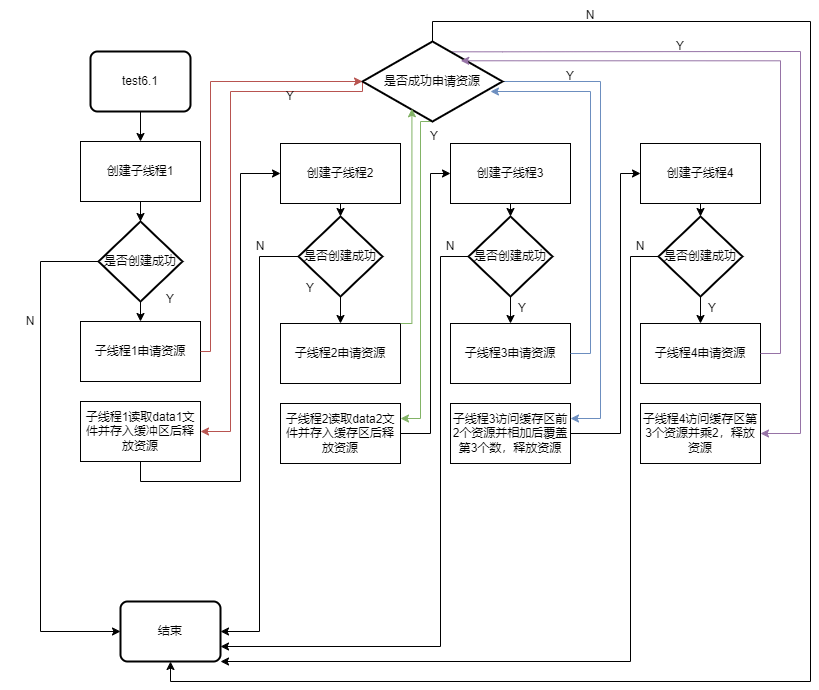
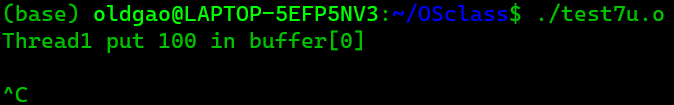
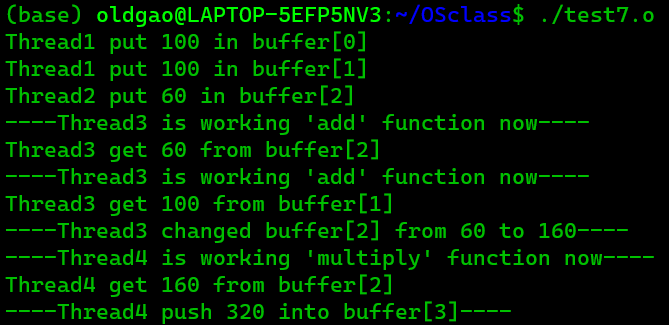


图29.test7流程图

# 实验结果及分析

图30.test7运行结果 图31.资源不足

如图，实验结果符合实验预期。那么如果使信号b设置为1，使得资源不足以线程1和2进行数据写入会发生什么？上图31.为实验结果，可以发现程序会一直等待资源，但是由于此时程序不可能等待到资源，因此会陷入死循环，只能手动终止。

# 运行结果与感想

运行结果如上图30.所示。

通过本次实验，我巩固了多线程编程的知识，并且将课程中所学的PV操作落实于实践，使得抽象的知识一下子变得具象起来，实验过后，我对PV操作的理解加深，并且在申请缓存区和申请mutex的顺序谁先谁后的问题上，从单纯记顺序转变为靠概念和原理来推理顺序。

# 参考资料

1. OS实验指导书