操作系统
实验指导书

实验报告要求

（1）计算机学院学生只针对实验四和实验五撰写实验报告；辅修学生只针对实验五撰写实验报告。格式内容按照公布的实验报告模板制作。

（2）实验报告只需将 PDF 电子版提交到课堂派。若出错，请联系老师打回重交。文件名为：学号中文姓名-实验报告.pdf。

（3）所有实验源代码通过课堂派打包提交。文件名为：学号中文姓名-源代码.zip。

实验要求

（1）本实验共 6 个实验。针对**计算机学院**的学生，**实验 1，2，3，4，5 为必做实验，其中实验 1 为熟悉系统，实验 6 为选作实验。实验 3、4 有选做题。**针对**辅修**的学生，**实验 1，2，3，5 为必做实验，选做题可做可不做**。

（2）老师会按照课程进度布置相关实验，请同学们尽量按照进度完成相关实验，以免期末突击。部分实验只需要将结果提交到课堂派即可得到分数。部分实验需要跟老师面对面检查。请同学关注课程进度，**按照与老师约定的时间进行实验代码的检查工作**。

（3）实验报告格式参照模板要求。

（4）**实验不允许抄袭。发现抄袭双方一律 0 分记。**请不要抄袭，也请自己保护好自己的代码。

实验环境说明

方法一：

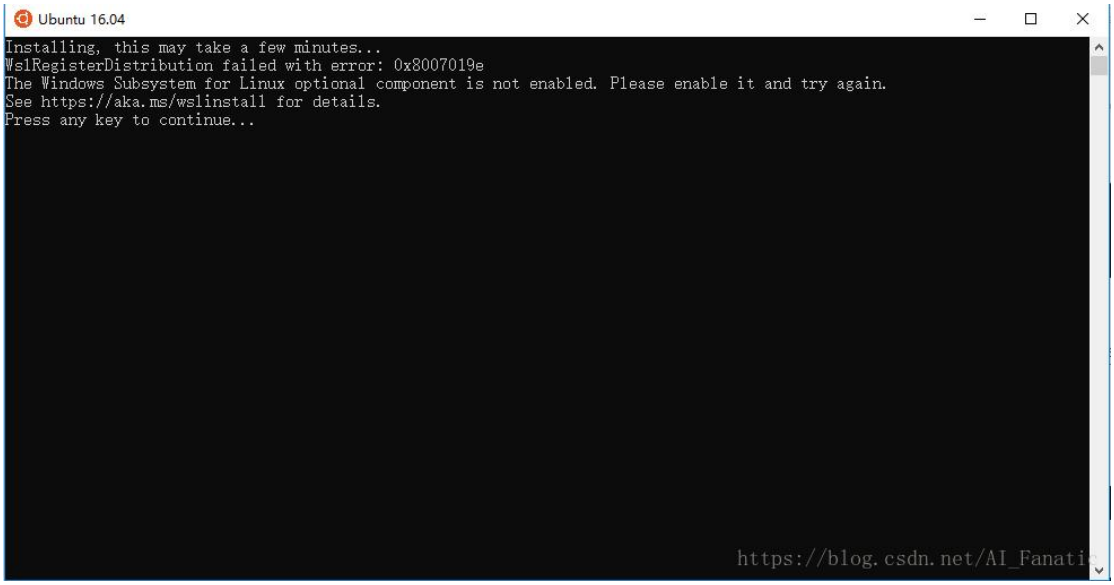
课程提供 12 学时的实验时间，地点在实训楼。实验中心的机器提供 Linux 操作系统，同学可以在此操作系统上进行程序的编写调试过程。由于实验时间不足以完成这么多实验任务，因此不建议大家只依靠实训楼的实验环境，因此强烈推荐方法二。

- 学校机器，用户名：student，密码：bjut-cs

方法二：

可以采用下面提到的两种方法其中的一种在自己的电脑上安装实验环境。

- **选择 1：在自己的机器上安装一个 Linux 虚拟机。虚拟机的安装步骤如下：**
  - 步骤一：在 VMWare 网站上下载一个 player
  - 步骤二：在 Ubuntu.com 上下载一个 Ubuntu Linux 的 iso，安装在此 Player 中（或者其他品牌的 Linux 也可以，均可以完成本实验）
  - 步骤三：安装过程中记住自己设置的用户名和密码，登陆的时候会使用。
  - **老师给出安装的详细文档，具体步骤参看该文档。**
- **选择 2：使用 Win10 中的 Linux 系统的方法：**
  - 在 Win10 的微软 microsoft store 中搜索（有好几个，老师装的是 Ubuntu 18.04LTS），并安装，然后启动它。
  - 第一步如果失败（如下图），则需要做如下设置：
    - 控制面板→程序→程序与功能→启动或关闭 Windows 功能→勾选“适用于 Linux 的 Windows 子系统”→重启 windows 系统



  - 安装过程中需要设置用户名和密码，请认真记录。在这个环境中可以熟悉实验 1 中各个指令。
  - 为了可以做其他实验，需要安装相关组件。步骤如下：
    - 编辑 sources.list

      sudo vim /etc/apt/sources.list

      【注意，vim 的使用方法可以在网上搜索。建议学会 vim 编辑器。】

- 设置国科大信息源

  按 i 进入编辑，在最前面加上下面的命令：

  deb https://mirrors.ustc.edu.cn/ubuntu/ xenial main restricted universe multiverse

  按 esc，然后:wq 保存
- 获取信息

  sudo apt-get update
- 安装相关组件

  sudo apt-get install build-essential

  安装时问是否继续，输入 Y

  然后就是等待安装完成
- 全部安装成功后，尝试写一个 hello world 程序来测试环境

实验一
UNIX/Linux 入门

实验学时：1 学时
实验类型：验证型

一、实验目的

1、了解 UNIX/Linux 运行环境；
2、熟悉 UNIX/Linux 的常用基本命令；
3、熟悉和掌握 UNIX/Linux 下 C 语言程序的编写、编译、调试和运行方法；
4、掌握如何在 C 语言程序中使用命令行参数。

二、实验内容

1、熟悉 UNIX/Linux 的常用基本命令。实验一的附录中列出了一些常见的 UNIX/Linux 基本命令，如 ls、who、pwd、ps 等，通过如下步骤进行尝试：

（1）在英文系统中找 Applications→Accessories →Terminal，并运行。在中文系统中，找 应用→附件→终端，并运行。Terminal/终端是一个命令行程序，会显示一个提示符，等待用户输入相关命令，回车之后执行。

（2）阅读第 4 项中介绍的命令，尝试运行，仔细阅读命令执行结果。由于命令的返回信息用英文来描述，需要大家耐心阅读。

2、熟悉和掌握 UNIX/Linux 下 C 语言程序的编写、编译、调试和运行方法。

在这里老师会带领大家编写一个简单的显示“Hello World!”C 语言程序，用 gcc 编译器编译并观察编译后的结果，然后运行它。步骤如下：

（1）选择一个目录下创建一个文件 example.c

（2）双击代表 example.c 的图标进入编辑器并输入 hello world 代码。代码如下：

```
#include “stdio.h”
void main()
{
    printf(“Hello World!\n”;
}
```

（3）保存并退出

（4）在终端（Terminal）中对 example.c 进行编译。编译命令为：

```
gcc example.c -o example
```

（5）运行编译好的程序。指令为：

```
./example
```

3、掌握如何在 C 语言程序中使用命令行参数。

在上一个任务中，为了使用 gcc 对我们编写的源程序进行编译，我们使用了下面的命令：

```
gcc example.c -o example
```

请注意，这里 gcc 是命令，而后面的“example.c –o example”则是这个命令的命令行参数。在使用 C 语言进行程序设计的时候，我们可以设计出一个应用程序，让它通过用户输入的命令行参数来得到相关信息。

下面给出一个带命令行参数的 C 语言程序的例子。假设这个程序的名字为 argtest.c：

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
   int i;
   printf("argc = %d\n", argc);
   for (i = 0; i <argc; i++) {
      printf("Argument %d: %s\n", i, argv[i]);
   }
   return (EXIT_SUCCESS);
}
```

之后使用下面的命令编译它：

```
gcc argtest.c -o argtest
```

最后，在命令行窗口中执行下面的命令，并分析运行结果：

```
./argtest
./argtest hello
./argtest hello world
```

三、实验要求

根据实验内容熟悉 Linux 操作系统，尝试执行 Linux 命令，并编写 Hello World 程序。由于此实验属于验证型实验，只需将 Hello World 程序执行结果提交到课堂派指定位置。

四、Linux 常用命令

1、Linux 系统常用命令格式：

```
  command  [option]  [argument1]  [argument2]  ...
```

其中 option 以“-”开始，多个 option 可用一个“-”连起来，如“ls -l -a” 与“ls -la”的效果是一样的。根据命令的不同，参数分为可选的或必须的；所有的命令从标准输入接受输入，输出结果显示在标准输出，而错误信息则显示在标准错误输出设备。可使用重定向功能对这些设备进行重定向。

命令在正常执行结果后返回一个 0 值，如果命令出错可未完全完成，则返回一个非零值(在 shell 中可用变量$?查看)。 在 shell script 中可用此返回值作为控制逻辑的一部分。命令如下：

```
echo $?
```

2、在使用 Linux 命令时的注意事项

（1）Linux 系统中，所有的命令、文件名、目录名都是大小写敏感的；

（2）命令与参数之间、参数与参数之间需要有明确的空格分开。

3、Linux 中常用的一些帮助命令

Linux 系统中为了让用户可以方便地使用命令，提供了几个帮助命令来让用户在不熟悉某些命令的含义或者相关信息的时候可以进行查询。下面介绍这些命令。

（1）which 命令

如果希望知道一个命令在文件系统的什么位置，可以使用“which”命令。

例如，想要知道 gcc 在文件系统的位置，可以在终端窗口键入下面的命令：

```
which gcc
```

（2）man 命令

man 命令用于获取相关命令的帮助信息。

例如想知道 ls 的帮助信息，可以在终端窗口键入下面的命令：

```
man ls
```

（3）info 命令

info 命令用于获取相关命令的详细使用方法。

例如想知道 ls 的详细使用方法，可以在终端窗口键入下面的命令：

```
info ls
```

4、应该熟练使用的一些命令

表 1 中列出了大家应该熟练掌握的命令，请详细阅读命令解释，结合上面提到的帮助命令进行练习。

表 1 常用命令

命令	命令解释
ls	列出文件和目录的清单
ls -a	列出所有文件和目录的清单
ls -l	以详细方式列出文件和目录的清单。 在结果中可以看到每个文件/目录的访问权限、文件所有者、长度、时间、文件名等信息。
mkdir	创建一个目录。 例如，下面的命令在当前目录下创建一个目录 abc `mkdir abc`
cd *directory*	将当前目录改变为指定目录 *directory*
cd	将当前目录改变为主目录
cd ..	将当前目录改变为父目录
pwd	显示当前目录的路径
cp *file1 file2*	拷贝文件 file1，并将结果文件命名为 file2
mv *file1 file2*	移动（或重命名）文件 file1 为 file2
rm *file*	删除文件
rmdir *directory*	删除目录
cat *file*	显示一个文件（文件名为 *file*）
more *file*	显示文件，每次一页（一屏）。按空格键翻到下一页，按回车键向下滚动一行。
head *file*	显示一个文件的最初若干行
tail *file*	显示一个文件的最后若干行
grep *'keyword' file*	在一个文件中查找关键字的出现
wc *file*	计算一个文件中的行数、单词数、字符数

command > *file*	将 标准输出 重定向 到一个文件。
	例如，如果仅执行下面的命令：
	`ls`
	命令的结果----文件和目录清单 将被输出到 标准输出，即屏幕。
	如果执行
	`ls > list.txt`
	ls 命令的输出结果将不会显示在屏幕，而是被重定向到文件 list.txt，即文

	件和目录清单被保存到 `list.txt` 文件。

command >> *file*	把标准输出追加到一个文件。
	请同学进行一个对比实验。
	首先，执行下面两条命令：
	`ls > list1.txt`
	`ls > list1.txt`
	然后，执行下面两条命令：
	`ls > list2.txt`
	`ls >> list2.txt`
	最后，对比 `list1.txt` 与 `list2.txt` 有何不同。

command < *file*	把 标准输入 重定向 为一个文件。 执行一个命令时，默认的标准输入是键盘。输入重定向使得运行的程序从一个文件中读取输入，而不是等待键盘输入。**我们在调试/测试一个程序时可以利用输入重定向，这样，就不必辛辛苦苦地一遍一遍敲输入数据了。**
command1 | *command2*	管道，将 *command1* 的输出 作为 *command2* 的输入
cat *file1 file2* > *file0*	将 *file1* 和 *file2* 连接，结果保存在 *file0*
sort	排序数据
who	列出 当前**登录**用户 的清单
ps	列出当前各个进程。

==========END==========

实验二
进程创建与管道通信

实验学时：3 学时

实验类型：验证型、设计型

一、实验目的

1、加深对进程概念的理解，明确进程与程序的区别；
2、进一步认识并发执行的实质；
3、学习在 Linux 操作系统中父子进程之间进行管道通信的方法。

二、实验内容

1、学习使用 fork()系统调用创建子进程，体会父子进程之间的并发关系。

请编写一段程序，使用系统调用 fork()创建两个子进程，实现当此程序运行时，在系统中有一个父进程和两个子进程在活动。父进程的功能是输出一个字符“a”；两个子进程的功能是分别输出一个个字符“b”和一个字符“c”。

多次运行这个程序，试观察记录屏幕上的显示结果，并分析原因。

另外，为了更好地展示进程之间的父子关系，大家可以使用 getpid()系统调用来获取当前进程的 PID，并用 getppid()用于获取当前进程的父进程的 PID。

2、继续体会进程之间的并发关系

修改刚才的程序，将每一个进程输出一个字符改为用一个循环输出 1000 个字符（父进程输出 1000 个“a”，子进程分别输出 1000 个“b”和“c”），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。父进程使用系统调用 pipe()创建一个无名管道，二个子进程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出来。

补充材料中给出了管道通信实现过程中需要使用的系统调用的说明，请仔细阅读。

三、实验要求

1、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。任务 1 和任务 2 在课堂派上提交结果，任务 3 需要在全部实验完成后找老师一对一检查。
2、请在编写程序的过程中注意采用良好的书写风格。例如，**缩进的格式、函数变量名的命名要有明确的意义，在执行系统调用后需要立即检查返回值进行出错处理**。
3、对于各种不明白的系统调用，请利用 man 命令的帮助，以及可靠的网络资源。

四、补充材料

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。在进行管道的实验之前请再次参考课程 PPT 中对于管道部分的描述，并仔细研究示例代码。

我们这个实验中使用的是无名管道，会用到下面的三个系统调用。

1、 无名管道的创建

无名管道由 pipe（）函数创建。要使用这个系统调用需要引用头文件 unistd.h。具体使用方法如下：

```
#include <unistd.h>
int pipe(int filedis[2]);
```

对于 pipe 系统调用而言，参数 filedis 是一个长度为 2 的整数数组，表示了两个文件描述符：filedis[0]专门为读操作服务，filedis[1]专门为写操作服务。

2、从管道中读数据

从管道中读数据需要使用 read 系统调用。read 系统调用不仅可以实现管道的读，还可以实现文件的读操作。由于在 Linux 中，将管道看成是一个虚拟文件，因此完全没有必要为管道读编制新的系统调用。read 系统调用的格式如下：

```
int read(int fd, char *buf, unsigned nbyte)
```

功能：从 fd 所指示的文件中读出 nbyte 个字节的数据，并将它们送至由指针 buf 所指示的缓冲区中。返回值为一个整数，表示读入了多少字节。如该文件被加锁，则等待，直到锁打开为止。

在进行管道读的时候，将 fd 设置为已经创建管道的读口即可。

3、向管道中写数据

向管道中写数据使用 write 系统调用。同样，write 系统调用不仅可以实现向管道的写，也可以实现向文件的写操作。write 系统调用格式如下：

```
int write(int fd, char *buf, int nbyte)
```

功能：把 nbyte 个字节的数据从 buf 所指向的缓冲区写到由 fd 所指向的文件中。如文件加锁，暂停写入，直至开锁。

在进行管道写的时候，将 fd 设置为已经创建管道的写口，并把要写入的数据事先放置在 buf 中，之后就可以使用这个系统调用将数据写入。

=========END=========

实验三
线程管理

实验学时：2 学时
实验类型：设计

一、实验目的

1、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；
2、掌握向线程传递参数的方法。

二、实验内容

1、 创建线程。

在这个任务中，需要在主程序中创建两个线程 myThread1 和 myThread2，每个线程打印一句话。**提示：**先定义每个线程的执行体，然后在主函数中使用 pthread_create 负责创建两个线程。整个程序等待子线程结束后再退出。

2、 向线程传递参数。

在上一个程序的基础上，分别向两个线程传递一个字符和一个整数，并让线程负责将两个参数的值打印出来。

3、 **【选作题】使用两个线程实现数组排序。**

主程序中用数组 data[1000]保存 1000 个整数型数据（赋值为 1 到 10，循环 100 次），创建两个线程，一个线程将这个数组中的数据从大到小排列输出；另一个线程求出数组中所有数据的和。

三、实验要求

1、按照要求编写程序，放在相应的目录中，编译成功后执行。任务 1 和任务 2 的实验结果请提交到课堂派的相应位置。如果做了选做题，需要在全部实验完成后找老师一对一检查。
2、请在编写程序的过程中注意采用良好的书写风格。例如，**缩进的格式、函数变量名的命名要有明确的意义，在执行系统调用后需要立即检查返回值进行出错处理**。
3、对于各种不明白的系统调用，请利用 man 命令的帮助，以及可靠的网络资源。

四、补充材料

1、 pthread 库

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，以保可以使用 pthread 库。由于 pthread 库不属于 linux 系统库，所以在进行程序编译时要加上-lpthread 选项，否则编译不会通过。

例如，假设源程序名字是 example.c，需要使用下面的编译命令将此源程序编译为目标程序 example：

```
gcc example.c -lpthread -o example
```

2、 函数 pthread_create()的用法

函数 pthread_create()用来创建一个线程，它的原型为：

```
int  pthread_create(pthread_t  *  thread, pthread_attr_t * attr, void *
      (*start_routine)(void *), void * arg);
```

第一个参数为指向线程标识符的指针，第二个参数用来设置线程属性，第三个参数是线程运行函数的起始地址，最后一个参数是运行函数的参数。这里，如果所创建的函数 thread 不需要参数，则最后一个参数设为空指针。第二个参数一般也设为空指针，这样将生成默认属性的线程。

当创建线程成功时，函数返回 0，若不为 0 则说明创建线程失败，常见的错误返回代码为 EAGAIN 和 EINVAL。EAGAIN 表示系统限制创建新的线程，在当前情况下已经不能再创建线程（即线程数目过多了）；EINVAL 表示第二个参数代表的线程属性值非法。

创建线程成功后，新创建的线程则运行第三个参数和第四个参数所确定的函数。

下面给出一些例子代码。

（1）声明和创建线程名为 myThread1，不传递参数的代码片段：

```
int ret;
pthread_t id1,id2;
ret = pthread_create(&id2, NULL, (void*)myThread1, NULL);
if (ret)
{
    printf("Create error!\n");
}
```

（2）声明和创建线程名为 myThread1，且向其传递一个整型参数的代码片段：

```
int ret;
pthread_t tidp;
int test=4;
int *attr=&test;
ret=pthread_create(&tidp,NULL,myThread1,(void *)attr);
if (ret)
{
    printf("Create error!\n");
}
```

（3）可以接受一个整数参数的线程 Mythread1 的代码片段：

```
void *MyThread(void *arg)
{
    int *num;
    num=(int *)arg;
    printf("%d \n",*num);
    return (void *)0;
}
```

3、 函数 pthread_join()和函数 Pthread_exit()的用法

函数 pthread_join()用来等待一个线程的结束。函数原型为：

```
int pthread_join(pthread_t th, void **thread_return);
```

第一个参数为被等待的线程标识符，第二个参数为一个用户定义的指针，它可以用来存储被等待线程的返回值。这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源被收回。

一个线程的结束有两种途径，一种是像上面的例子一样，函数结束了，调用它的线程也就结束了；另一种方式是通过函数 pthread_exit 来实现。它的函数原型为：

```
void pthread_exit(void *retval);
```

唯一的参数是函数的返回代码 retval。

这两个函数在配合使用的时候，只要 pthread_join 中的第二个参数 thread_return 不是 NULL，retval 的值将被传递给 thread_return。

另外，一个线程不能被多个线程等待，否则第一个接收到信号的线程成功返回，其余调用 pthread_join 的线程则返回错误代码 ESRCH。

=========END=========

实验四
利用信号量实现线程互斥与同步

实验学时：4 学时

实验类型：验证、设计型

一、实验目的

1、学习 UNIX 类（System V）操作系统信号量机制；

2、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。

二、实验内容

1、生产者消费者问题。

有数据文件 1.dat 和 2.dat 分别存放了 10 个整数。创建 4 个线程，其中两个线程 read1 和 read2 负责分别从文件 1.dat 和 2.dat 中读取一个整数到公共的缓冲区，另两个线程 operate1 和 operate2 分别从缓冲区读取数据作加运算和乘运算。使用信号量控制这些线程的执行，保证缓冲区中的数据只能被计算一次（加或者乘），计算完成之后才能继续进行数据的读取工作。

提示：这是一个缓冲区长度为 2 的生产者消费者问题。read1 和 read2 是生产者，operate1 和 operate2 是消费者。需要互斥使用的缓冲区的长度为 2。在进行读取的时候不见得一定是 read1 读一个数然后 read2 读一个数，允许 read1 读得快（或者 read2 读得快），所以缓冲区中当前的数据都来自于 1.dat 或者 2.dat 是被允许的。计算也可能出现连续加和连续乘的情况。

例如，事先编辑好数据文件 1.dat 和 2.dat 的内容分别为 1 2 3 4 5 6 7 8 9 10 和 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10，那么运行你编写的程序，可能得到如下类似的结果：

```
1+-1=0
-2*2=-4
3*4=12
-3*-4=12
5+-5=0
6*-6=-36
…
```

建议预先写出伪代码描述算法，再进行程序设计实现。

2、【选作题】如果严格限制加和乘的两个操作数必须分别来自 1.dat 和 2.dat，且加法和乘法要严格交叉工作，应该如何修改上面的程序？

运行结果为：

```
1+-1=0
2*-2=-4
-3+3=0
4*-4=-16
5+-5=0
-6*6=-36
…
```

三、实验要求

1、按照要求编写程序，放在相应的目录中，编译成功后执行，两个任务都需要在全部实验完成后找老师一对一检查，并写出实验报告。
2、为保证程序的正确性，请先按照课堂上的方法设计出算法，再进行程序的编制工作。
3、请在编写程序的过程中注意采用良好的书写风格。例如，**缩进的格式、函数变量名的命名要有明确的意义，在执行系统调用后需要立即检查返回值进行出错处理。**
4、对于各种不明白的系统调用，请利用 man 命令的帮助，以及可靠的网络资源。

四、补充材料

Linux 中的信号量本质上是一个非负的整数计数器，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。函数 sem_wait()减少信号量。下面我们逐个介绍和信号量有关的一些函数，它们都在头文件 /usr/include/semaphore.h 中定义。因此编写信号量的程序，需要在程序开始位置引入这个头文件。代码如下：

```
#include “semaphore.h”
```

1、函数 sem_init()的用法

信号量的数据类型为结构 sem_t，它是一个长整的数。函数 sem_init（）用来初始化一个信号量。它的原型为：

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

其中，sem 为指向信号量结构的一个指针；pshared 不为 0 时此信号量在进程间共享，否则只能为当前进程的所有线程共享；value 给出了信号量的初始值。

2、函数 sem_post()的用法

函数 sem_post()用来增加信号量的值。当有线程阻塞在这个信号量上时，调用这个函数会使其中的一个线程不再阻塞，选择激活哪一个线程是由线程的调度策略决定的。这个函数的原型是：

```
int sem_post(sem_t * sem);
```

其中，sem 为指向信号量结构的一个指针。

3、函数 sem_wait()的用法

函数 sem_wait()被用来减少一个信号量的值。如果这个信号量的值大于 0，则这个信号量的值减 1。如果这个信号量的值为 0，则阻塞当前线程。这个函数的原型是：

```
int sem_wait(sem_t * sem);
```

其中，sem 为指向信号量结构的一个指针。

=========END=========

实验五
基于消息队列和共享内存的进程间通信

实验学时：2 学时

实验类型：验证、设计型

一、实验目的

1、了解和熟悉 Linux 支持的消息通信机制及其使用方法

2、了解和熟悉 Linux 系统的共享存储区的原理及使用方法。

二、实验内容

1、消息的创建、发送和接收

主函数中使用 fork()系统调用创建两个子进程 sender 和 receiver。sender 负责接收用户输入的一个字符串，并构造成一条消息，传送给 receiver。Receiver 在接收到消息后在屏幕上显示出来。此过程一直继续直到用户输入 exit 为止。在程序设计过程中使用 msgget()、msgsnd()、msgrcv()、msgctl()。

2、共享存储取得创建、附接和断接

主函数中使用 fork()系统调用创建两个子进程 sender 和 receiver。Sender 创建共享内存区域并从用户输入得到一个整数放入共享内存区域，receiver 负责取出此数并将此整数的平方计算出来。要求程序可以计算 10 个整数的平方值。在程序设计过程中使用 shmget()、shmat()、shmctl()。

三、实验要求

1、两个任务二选一完成即可。用 WSL 做实验的同学（即在 windows 的应用市场里装了 Ubuntu），有可能消息队列创建不成功，若这样可选“共享内存”任务。

2、按照要求编写程序，放在相应的目录中，编译成功后执行，针对任务 1 或任务 2 按照要求分析执行结果，并写出实验报告。

3、请在编写程序的过程中注意采用良好的书写风格。例如，**缩进的格式、函数变量名的命名要有明确的意义，在执行系统调用后需要立即检查返回值进行出错处理**。

4、对于各种不明白的系统调用，请利用 man 命令的帮助，以及可靠的网络资源。

四、补充材料

1、消息队列

（1）创建/获取消息队列 msgget

创建/获取消息队列 msgget()的功能是创建或者访问一个消息队列。其原型为：

```
int msgget(key_t key, int msgflg);
```

该函数使用头文件如下：

```
#include <sys/ types.h>
#include <sys/ ipc.h>
#include <sys/ msg.h>
```

参数说明：

key：消息队列关联的键。

msgflg：消息队列的建立标志和存取权限。如果 flag 为 IPC_CREAT，表示如果内核中

没有此队列，则创建它。Flag 为 IPC_EXCL 和 IPC_CREAT 时，表示如果队列已经存在，则 msgget 执行失败。IPC_EXCL 单独使用是没有用处的。

此函数的返回值是创建的消息队列的标志符。

下面是一个创建和打开消息队列的例子。注意，第 3 行的 0666 是一个八进制整数（如果以二进制表示，是 110 110 110），它表示了所创建的消息队列的访问权限，分别针对所有者（owner）、所有者的同组（group）和其他（other）对该消息队列“可读”、“可写”。在创建 System V IPC 中的消息队列、信号量集、共享存储区时，都需要指定访问权限。

```
int open_queue(int keyval)
{
    int qid;
    qid = msgget(keyval, IPC_CREAT | 0666);
    if (qid == -1) {
        perror("Failed in calling msgget");
        return (-1);
    }
    return (qid);
}
```

（2）消息发送 msgsnd

其功能是发送一条消息到指定的消息队列。此函数的原型是：

```
int msgsnd(msgid, msgp, size, flag)
```

该函数使用头文件如下：

```
#include <sys/ types.h>
#include <sys/ ipc.h>
#include <sys/ msg.h>
```

参数说明：

msgid：消息队列描述符；

msgp：是指向用户存储区的一个构造体指针；

size：指示由 msgp 指向的数据结构中字符数组的长度，即消息的长度。

flag：规定当内核用尽了内部缓冲空间时应执行的动作。若在标志 flag 中未设置 IPC_NOWAIT 位，则当该消息队列中的字节数超过最大值，或系统范围的消息数超过某一最大值时，调用 msgsnd()的进程进入睡眠。若是设置为 IPC_NOWAIT，则在此情况下，msgsnd()立即返回。

msgsnd 的返回值为 0 表示正确执行，否则为-1 表示出错。

为了实现消息发送，进程必须对消息队列有 write 权限。

为了实现消息接收，进程必须对消息队列有 read 权限。

（3）消息接收 msgrcv

其功能是从指定消息队列接收一条消息。此函数的原型是：

```
ssize_t msgrcv(msgid, msgp, size, type, flag)
```

参数说明：

msgid：消息队列描述符。

msgp：用来存放接收到的消息的用户数据结构的地址。

size：指示由 msgp 指向的数据结构中字符数组的长度，即消息的长度。

type：用户要读的消息类型：

type 为 0 ：接收该队列中的全部消息；

type 为正：接收队列中第一个类型为 type 的消息；

type 为负：接收小于或等于 type 绝对值的最小的类型的第一个消息。

flag：用来指明在队列没有数据的情况下 msgrcv()所应采取的行动。如果是 IPC_NOWAIT，那么如果队列中没有可读的消息，则不等待，直接返回 ENOMSG 错误。如果是 MSG_NOERROR，则当消息超过了 size 的时候会被截断而不会报错。当 msgflg 为 MSG_EXCEPT 时，如果 type 的值大于 0，则接收类型不等于 type 的第一条消息。

msgrcv 返回收到的消息正文的字节数。

（4）消息控制 msgctl

其功能是消息控制操作。此函数的原型是：

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

该函数使用头文件如下：

```
#include <sys/ types.h>
#include <sys/ ipc.h>
#include <sys/ msg.h>
```

参数说明：

msqid：消息队列标识码。

cmd：操作命令，可能值如下面：

IPC_STAT：将 msqid 所指定的消息队列的信息拷贝一份到 buf 指针所指向的地址。调用者必须对消息队列有读权限。

IPC_SET：将由 buf 所指向的 msqid_ds 结构的一些成员写入到与这个消息队列关联的内核结构。同时更新的字段有 msg_ctime。

IPC_RMID：删除指定的消息队列，唤醒所有等待中的读者和写者进程。

2、共享内存

（1）共享存储区的建立 shmget

函数原型为：

```
int shmget(key_t key, size_t size, int shmflg);
```

其功能是建立（获得）一块共享存储区，返回该共享存储区的描述符 shmid；若尚未建立，便为进程建立一个指定 size 大小的共享存储区。如果是一个新的共享存储区，shmget()将初始共享存储区中所有单元设置为 0。

该函数使用头文件如下：

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

参数说明：

key：共享储存区描述符。

size：共享储存区的大小。

shmflg：建立标志和储存权限，取值如下，且可通过 or 组合在一起：

IPC_CREAT：建立新的共享区段。

IPC_EXCL：和 IPC_CREAT 标志一起使用，如果共享区段已存在失败返回。

SHM_HUGETLB：使用"huge pages"来分配共享区段。

SHM_NORESERVE：不要为共享区段保留交换空间。

（2）共享存储区的控制 shmctl

函数原型为：

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

其功能是对共享存储区 shmid 执行操作 cmd，对其状态信息进行读取和修改。

该函数使用头文件如下：

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

参数说明：

shmid：共享存储区描述符。

cmd：指定下列 5 种命令中一种，使其在 shmid 指定的共享存储区上执行：

IPC_STAT：对此段取 shmid_ds 结构，并存放在由 buf 指向的结构中。

IPC_SET：按 buf 指向的结构中的值设置与此段相关结构中的下列三个字段：shm_perm.uid、shm_perm.gid 以及 shm_perm.mode。

IPC_RMID：从系统中删除该共享存储区。因为每个共享存储区有一个连接计数（shm_nattch 在 shmid_ds 结构中），所以除非使用该段的最后一个进程终止或与该段脱接，否则不会实际上删除该存储段。不管此段是否仍在使用，该段标识符立即被删除，所以不能再用 shmat 与该段连接。

SHM_LOCK：锁住共享存储段。此命令只能由超级用户执行。

SHM_UNLOCK：解锁共享存储段。此命令只能由超级用户执行。

（3）共享存储区的附接 shmat

在进程已经建立了共享存储区或已获得了其描述符后，还须利用系统调用 shmat()将该共享存储区附接到用户给定的某个进程的虚地址上，并指定该存储区的访问属性，即指明该区是只读，还是可读可写。此共享存储区便成为该进程虚地址空间的一部分。

函数原型为：

```
addr = shmat(shmid,addr,flag)
```

该函数使用头文件如下：

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

参数说明：

shmid：共享存储区描述符；

addr：用户给定的，将共享存储区附接到进程的虚地址；

flag：规定共享存储区的读、写权限，以及系统是否应对用户规定的地址做舍入操作。其值为 SHM_RDONLY 时，表示只能读；其值为 0 时，表示可读、可写；其值为 SHM_RND（取整）时，表示操作系统在必要时舍去这个地址。

返回值是共享存储区所附接到的进程虚地址 addr。

（4）共享存储区的断开 shmdt

当进程不再需要一个共享存储段时，可以调用 shmdt()将它从进程的地址空间分离。

其函数原型如下：

```
int shmdt(void *addr);
```

该函数使用头文件如下：

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

参数说明：

addr：是以前调用 shmat()时的返回值。注意：这个函数仅是是一个共享存储区不再与调用进程的地址空间相连，它并不实际删除共享存储区本身。若想删除共享存储区，应该使用 shmctl（）函数，并设置其 cmd 参数为 IPC_RMID。

=========END=========

实验六
使用信号进行进程间通信

实验学时：2 学时

实验类型：验证、设计型

一、实验目的

1、学习操作系统信号机制的实现方法；

2、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信号的发送和接收的一般过程。

二、实验内容

1、学习任务。

学习补充材料中的UNIX信号及其描述。

2、使用信号进行进程间通信。

编写一个程序，完成下列功能：实现一个SIGINT、用户自定义信号的处理程序，注册该信号处理程序。主函数中创建一个子进程，令父子进程都进入等待状态。SIGINT、 用户自定义信号的处理程序完成的任务包括：（1）打印接收到的信号的编号（2）打印进程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，并解释其含义。

三、实验要求

1、按照要求编写程序，放在相应的目录中，编译成功后执行，针对任务 2 按照要求分析执行结果。本任务是选作任务，若做了，需要在全部实验完成后找老师一对一检查。

2、请在编写程序的过程中注意采用良好的书写风格。例如，**缩进的格式、函数变量名的命名要有明确的意义，在执行系统调用后需要立即检查返回值进行出错处理**。

3、对于各种不明白的系统调用，请利用 man 命令的帮助，以及可靠的网络资源。

四、补充资料

1、UNIX 信号及其描述

信号（signal）是 UNIX 提供的进程间通信与同步机制之一，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。

接收进程何时处理信号？仅在进程被唤醒运行，或者它将要从一个系统调用返回的时候，进程才会处理信号。进程可以对信号做出哪些反应？进程可以执行某些默认动作（如终止运行），或者执行一个信号处理函数，或者忽略那个信号。UNIX 信号及其描述如表 2 所述。注意，不同版本的 UNIX 类操作系统支持的信号种类和编号可能略有不同。更详细的信息请查阅不同操作系统的相关手册。

表 2 UNIX 信号及其描述

信号编号	信号名称	描述

01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work 连接断开
02	SIGINT	Interrupt 当用户在终端上敲中断键（通常为DELTE或Cntl + C）时，由终端驱动程序发出。该信号发送给前台进程组中的所有进程。
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump 退出；当用户在终端上敲退出键（通常为Cntl + \）时，由终端驱动程序发出，终止前台进程组中的所有进程并生成**core**
04	SIGILL	Illegal instruction 执行了无效的硬件指令
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing 跟踪陷阱；触发执行用于跟踪的代码
06	SIGIOT	IOT instruction 由具体实现定义的硬件错误
07	SIGEMT	EMT instruction 由具体实现定义的硬件错误
08	SIGFPE	Floating-point exception 算数异常
09	SIGKILL	Kill; terminate process 终止进程
10	SIGBUS	Bus error 由具体实现定义的硬件错误。通常为某种类型的存储器错误
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space 无效的内存引用；进程试图访问其虚拟地址空间以外的位置
12	SIGSYS	Bad argument to system call 无效的系统调用
13	SIGPIPE	Write on a pipe that has no readers attached to it 如果向一个管道写，但是其读者已经终止，则产生此信号
14	SIGALRM	Alarm clock; issued when a process wishes to receive a signal after a period of time 时钟信号
15	SIGTERM	Software termination 终止信号，默认情况下由kill(1)命令发出
16	SIGUSR1	User-defined signal 1 用户定义信号1
17	SIGUSR2	User-defined signal 2 用户定义信号2
18	SIGCHLD	Death of a child 子进程消亡
19	SIGPWR	Power failure 电源故障

2、注册信号处理程序

注册信号处理程序需要使用 signal 系统调用。

signal 系统调用的原型为：

```
sighandler_t signal(int signum, sighandler_t handler);
```

该函数使用头文件如下：

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

参数说明：

signum：信号的编号，即表 2 中的信号编号列中的某个值，代表相关信号

handler：为下列 3 种情况之一：

常量 SIG_IGN，表示告诉系统忽略这个信号；

常量 SIG_DFL，表示信号发生是采取默认动作；

信号处理函数地址（函数指针），当信号发生时调用该函数。

signal()函数的返回值就是信号处理函数的地址，但如果出错，则返回 SIG_ERR。

因此，在编写此类程序的时候，首先需要编写一个信号处理函数。此函数要求有一个 int 型参数，其返回值类型为 void。之后在主函数中调用 signal 将此信号处理函数注册为与某个信号相关。因此在程序执行的时候，一旦发生了此信号，就会执行信号处理函数的函数体。

下面给出一个程序例子来说明此类应用程序的编写方法：

```
#include <stdio.h>
#include <signal.h>
static void sig_usr(int signo)     /* argument is signal number */
{
    if (signo == SIGUSR1)
        printf("received SIGUSR1\n");
    else
    {
        printf("received signal %d\n", signo);
        exit(1);
    }
}
int main(void)
{
    if (signal(SIGUSR1, sig_usr) == SIG_ERR)
    {
        printf("can't catch SIGUSR1\n");
        exit(1);
    }
    for ( ; ; )
        pause();
}
```

将上述程序编译，假设可执行文件为 a.out，通过下面的命令行程序执行这个应用程序：

```
./a.out &
```

执行上述命令后，终端窗口中首先显示该进程的进程 ID，随后显示命令提示符。假设

其 PID 为 7216。依次执行下列命令，试解释命令的含义和执行结果：

```
kill -USR1 7216
kill 7216
```

同学们可以使用 man 命令来学习 kill 系统调用的相关功能。

=========END=========