操作系统课程设计实验报告

实验题目：实验三 Linux 进程管理

姓　　名：张孜远

学　　号：20151521

组　　号：04

专　　业：卓越学院 智能计算与数据科学

班　　级：20186211

老师姓名：任彧老师

日　　期：2022 年 12 月 20 日

目　　录

一 题目介绍.....1
二 实验内容与思路.....2
三 遇到问题及解决方法.....2
四 核心代码及实验结果展示.....2
五 个人实验改进与总结.....5
5.1 个人实验改进.....5
5.2 个人实验总结.....5
六 参考文献.....6

一 题目介绍

实验目的：

1、通过对 Linux 进程控制的相关系统调用的编程应用，进一步加深对 Linux 进程概念的理解，明确进程和程序的联系与区别，理解进程并发执行的具体含义；

2、通过对 Linux 管道通信机制、消息队列通信机制、共享内存通信机制的应用，加深对不同类型的进程通信方式的理解；

3、通过对 Linux 的 Posix 信号量及 IPC 信号量的应用，加深对信号量同步机制的理解。

实验内容：

（1）实现一个模拟的 shell：

编写三个不同的程序 cmd1.c，cmd2.c，cmd3.c，每个程序的功能自定，分别编译成可执行文件 cmd1，cmd2，cmd3。然后再编写一个程序，模拟 shell 程序的功能，能根据用户输入的字符串（表示相应的命令名），去为相应的命令创建子进程并让它去执行相应的程序，而父进程则等待子进程结束，然后再等待接收下一条命令。如果接收到的命令为 exit，则父进程结束；如果接收到的命令是无效命令，则显示 Command not found ，继续等待。

（2）实现一个管道通信程序：

由父进程创建一个管道，然后再创建 3 个子进程，并由这三个子进程利用管道与父进程之间进行通信：子进程发送信息，父进程等三个子进程全部发完消息后再接收信息。通信的具体内容可根据自己的需要随意设计，要求能试验阻塞型读写过程中的各种情况，测试管道的默认大小，并且要实现进程间对管道的互斥访问。运行程序，观察各种情况下，进程实际读写的字节数以及进程阻塞唤醒的情况。

（3）利用 linux 的消息队列通信机制实现两个线程间的通信：

编写程序创建三个线程：sender1 线程、sender2 线程和 receive 线程，三个线程的功能描述如下：

① sender1 线程：运行函数 sender1()，它创建一个消息队列，然后，等待用户通过终端输入一串字符，将这串字符通过消息队列发送给 receiver 线程；可循环发送多个消息，直到用户输入 exit 为止，表示它不再发消息，最后向 receiver 线程发送消息 end1 ，并且等待 receiver 的应答，等到应答消息后，将接收到的应答信息显示在终端屏幕上，结束程序的运行。

② sender2 线程：运行函数 sender2()，它创建一个消息队列，然后，等待用户通过终端输入一串字符，将这串字符通过消息队列发送给 receiver 线程；可循环发送多个消息，直到用户输入 exit 为止，表示它不再发消息，最后向 receiver 线程发送消息 end2 ，并且等待 receiver 的应答，等到应答消息后，将接收到的应答信息显示在终端屏幕上，结束程序的运行。

③ receiver 线程运行 receive()，它通过消息队列接收来自 sender1 和 sender2 两个线程的消息，将消息显示在终端屏幕上，当收到内容为 end1 的消息时，就向 sender1 发送一个应答消息 over1 ；当收到内容为 end2 的消息时，就向 sender2 发送一个应答消息 over2 ；消息收完后删除消息队列。使用合适的信号量机制实现三个线程之间的同步与互斥。

（4）利用 linux 的消息队列通信机制实现两个线程间的通信：

编写程序创建两个线程：sender 线程和 receive 线程，其中 sender 线程运行函数 sender()，它创建一个消息队列，然后，循环等待用户通过终端输入一串字符，将这串字符通过消息队列发送给 receiver 线程，直到用户输入 exit 为止；最后，它向 receiver 线程发送消息 end ，并且等待 receiver 的应答，等到应答消息后，将接收到的应答信息显示在终端屏幕上，删除相关消息队列，结束程序的运行。receiver 线程运行 receive()，它通过消息队列接收来自 sender 的消息，将消息显示在终端屏幕上，直至收到内容为 end 的消息为止，此时，它向 sender 发送一个应答消息 over ，结束程序的运行。使用 无名信号量 实现两个线程之间的同步与互斥。

二 实验内容与思路

实验内容：

1、模拟 shell：基本功能 + find、grep 命令，并显示相应结果；

2、管道通信：基本功能 + 有名管道通信（独立进程）；

3、共享内存：基本功能 + 双向通信；

三 遇到问题及解决方法

遇到的问题：

1、实验编译可执行文件失败：

解决方法：编译时使用 gcc xxx.c -pthread 指令，添加参数-pthread 即可成功生成可执行文件；

2、在线程未创建时，程序已在 main 函数中结束：

解决方法：添加阻塞进程，调用线程函数 pthread_join；

实验方法：

在虚拟机中 Linux 环境下使用 C 语言进行编程，参照书本与课堂要求完成 Linux 进程管理中的三个实验。

四 核心代码及实验结果展示

实验过程和结果：

1、 模拟 shell：

1.1 使用 make 指令，生成 cmd1.c、cmd2.c、cmd3.c 和 shell.c 的可执行文件；

```
zhangziyuan@ubuntu:~/exp3/shell$ make
gcc cmd1.c -o cmd1
gcc cmd2.c -o cmd2
gcc cmd3.c -o cmd3
gcc shell.c -o shell
```

1.2 使用./shell 指令，进入 shell.c 的可执行文件（类比为 main 函数），验证功能；

1）分别进入 cmd1.c、cmd2.c 和 cmd3.c 的可执行文件 cmd1、cmd2 和 cmd3，正确输出 cmd1.c、cmd2.c 和 cmd3.c 中的内容；

2）使用 find 指令，在当前目录下寻找文件，指令格式为：find “file”；同时如若该文件不存在，则输出报错“find: ‘file’: No such file or dictionary”;

3）使用 grep 指令，在某个文件中寻找特定字段，指令格式为：grep string file；

4）使用 exit 指令，退出模拟 shell 实验；

```
zhangziyuan@ubuntu:~/exp3/shell$ ./shell
$ cmd1
This is cmd1.c file, which contains a word 'cmd1'.
$ cmd2
This is a cmd2.c file, which contains a word 'cmd2'.
$ cmd3
This is a cmd3.c file, which contains a word 'cmd3'.
$ find "cmd1.c"
cmd1.c
$ find "cmd2.c"
cmd2.c
$ find "cmd3.c"
cmd3.c
$ find "cmd.c"
find: ‘cmd.c’: No such file or directory
$ grep cmd1 cmd1.c
        printf("This is cmd1.c file, which contains a word 'cmd1'.\n");
$ grep cmd2 cmd2.c
        printf("This is a cmd2.c file, which contains a word 'cmd2'.\n");
$ grep cmd3 cmd3.c
        printf("This is a cmd3.c file, which contains a word 'cmd3'.\n");
$ exit
Simulating Shell Experiment exits
zhangziyuan@ubuntu:~/exp3/shell$
```

2、 管道通信：

1） 统计管道大小：

通过非阻塞方式将数据写入管道，每次写 1024 字节，即 1KB，直到管道满为止。

```
zhangziyuan@ubuntu:~/exp3/pip/count$ gcc count.c
zhangziyuan@ubuntu:~/exp3/pip/count$ ./count
Calulate the size of Pipline: 64KB
zhangziyuan@ubuntu:~/exp3/pip/count$
```

由上图可知，管道大小是 64KB；

2） 无名管道通信：

通过预编写的 pip.c 文件，创建三个子进程。分别向进程 1、进程 2 和进程 3 中写入数据“111”、“222”和“333”，验证父进程接收的数据情况。

```
zhangziyuan@ubuntu:~/exp3/pip/noname$ make
gcc pip.c -o pip -pthread
zhangziyuan@ubuntu:~/exp3/pip/noname$ ./pip
pid:2556 进程1写入数据：111
pid:2558 进程3写入数据：222
pid:2557 进程2写入数据：333
pid:2555 父进程接收数据:
111
222
333
zhangziyuan@ubuntu:~/exp3/pip/noname$
```

3）有名管道通信：

通过公共的头文件 share.h，编写 father.c、son1.c、son2.c 和 son3.c 源文件，通过 make 指令依次生成 father、son1、son2 和 son3 可执行文件。

```
zhangziyuan@ubuntu:~/exp3/pip/name$ make
gcc father.c -o father -pthread
gcc son1.c -o son1 -pthread
gcc son2.c -o son2 -pthread
gcc son3.c -o son3 -pthread
```

通过./father 指令先进入 father.c 的可执行文件，用于接受来自 son1、son2 和 son3 中的数据；通过./son1、./son2 和./son3 指令依次进入 son1.c、son2.c 和 son3.c 的可执行文件，分别在每个子进程中写入数据：this is son1、this is son2 和 this is son3，观察父进程接收到的数据情况，是否依次为 this is son1、this is son2 和 this is son3。

```
zhangziyuan@ubuntu:~/exp3/pip/name$ ./son1
pid:2819 进程1写入数据：this is son1
zhangziyuan@ubuntu:~/exp3/pip/name$
```

```
zhangziyuan@ubuntu:~/exp3/pip/name$ ./father
pid:2809 父进程接收数据:
this is son1
this is son2
this is son3
zhangziyuan@ubuntu:~/exp3/pip/name$
```

```
zhangziyuan@ubuntu:
File  Edit  View  Search  Terminal  Help
zhangziyuan@ubuntu:~/exp3/pip/name$ ./son2
pid:2829 进程2写入数据：this is son2
zhangziyuan@ubuntu:~/exp3/pip/name$
```

```
zhangziyuan@ubuntu:
File  Edit  View  Search  Terminal  Help
zhangziyuan@ubuntu:~/exp3/pip/name$ ./son3
pid:2839 进程3写入数据：this is son3
zhangziyuan@ubuntu:~/exp3/pip/name$
```

由上图可知，实验结果正确无误。

3、共享内存（双向通信）：

通过 make 指令依次生成 job1.c 和 job2.c 的可执行文件 job1 和 job2。在文件夹下打开两个终端，分别进入可执行文件 job1 和 job2。

先从 job1 中连续发送消息 4 次消息字符串“1”，观察 job2 中接收信息的情况；

再从 job2 中连续发送消息 5 次消息字符串“2”，观察 job1 中接收信息的情况；

最后在 job2 中输入 exit 指令，退出“共享”；同时在 job1 中输出相应提示信息：“对方终止会话”。

```
zhangziyuan@ubuntu:~/exp3/share$ make
gcc job1.c -o job1 -pthread
gcc job2.c -o job2 -pthread
zhangziyuan@ubuntu:~/exp3/share$ ./job1
发送：1
发送成功！
发送：1
发送成功！
发送：1
发送成功！
发送：1
发送成功！
发送：1
发送成功！
收到消息：2
收到消息：2
收到消息：2
收到消息：2
收到消息：2
对方终止会话
zhangziyuan@ubuntu:~/exp3/share$
```

```
zhangziyuan@ubuntu:~/exp3/share$ ./job2
收到消息：1
收到消息：1
收到消息：1
收到消息：1
收到消息：1
发送消息：2
发送成功！
发送：2
发送成功！
发送：2
发送成功！
发送：2
发送成功！
发送：2
发送成功！
发送：exit
zhangziyuan@ubuntu:~/exp3/share$
```

由上图可知，实验结果正确无误。

备注：我们也可以在 job1 中退出“共享”，如下图所示：

五 个人实验改进与总结

5.1 个人实验改进

项目实现创新点说明	1、根据实验指导书上的要求，以及老师课堂上提出的创新性要求，完成相应的实验工作。

5.2 个人实验总结

实验体会：

相较于之前的操作系统实验，本次实验更加复杂繁琐。我首先补充了大量有关消息队列和信号量机制的原理，牢固自己的理论知识；再通过阅读 Linux 内核源码，了解了相关 Linux 内核机制，再开始动手实践。

最开始参考 Github 上学长的代码，其逻辑是用一个 for 循环分别创建子进程，但由于其在接收信息后还会再打印一次数据，有 bug 但我一直没修好，所以后面就打算自己动手实践实现相应功能。

通过此次实验，我对于 Linux 的管道通信机制和共享内存机制的理解更深了。

管道通信可以轻松地实现两个进程之间的交互，在现实生活中可被用于多个场景（例如智能问答中的人机交互等），但其局限性也十分明显：该管道通信机制只能是单向的；其次如果发送的数据量过大，则会导致管道满和数据丢失的情况。

相较于管道通信机制，共享内存机制可以轻松实现多个进程之间的数据共享，即一个进程向共享内存中写入数据，一个进程向共享内存中读出数据，避免了系统调用的开销，但其局限性也十分明显：共享内存的容量是有限制的；当进程数量变多时，需要通过信号量合理设置同步机制，避免出现冲突的问题。

通过此次实验，我更加充分理解了消息队列和信号量机制的原理，让所学的理论应用于实际，成就感很高。

六 参考文献

[1] WEXITSTATUS 与 WIFEXITED：
https://blog.csdn.net/hit_shaoqi/article/details/53150890
[2] 有关消息队列 msgget()、msgsend()、msgrcv()、msgctl()的解释：
https://www.cnblogs.com/52php/p/5862114.html
[3] 有关 stderr()和 stdout()的解释：
https://www.cnblogs.com/mydomain/p/9817320.html
[4] 有关 pthread_join 的解释：
https://blog.csdn.net/yzy1103203312/article/details/80849831
[5] 有关 sem_init 的解释：
https://www.cnblogs.com/pipci/p/10179502.html