第二章 基本文件 I/O

1、书上实例练习
2、设计一个程序，实现将一个字符串“Hi, I'm a Linux programmer!”写入到文件 a.txt，然后将 a.txt 中前 10 个字符拷贝到文件 b.txt 中。

第三章　进程与线程

1、练习使用 fork 复制进程的例子，观察处理器的调度情况，尝试使用 sleep(1)来改变调度顺序。

2、练习 fork+execl/execv/execlp/execvp 的结合使用。

3、waitpid 实验　　waitpid.c
问题描述：
首先使用 fork 新建一个子进程，并使子进程暂停 5s（使用 sleep 函数），接下来，父进程使用 waitpid 函数，并使用参数 WNOHANG 使父进程不会阻塞。若有子进程退出，则 waitpid 返回子进程号；若没有子进程退出，则 waitpid 返回 0，并且父进程每隔一秒循环判断一次。

提示:
pr = waitpid(pid, NULL, WNOHANG)

4、多进程程序实验　　proc_expr1.c
实验目的：熟练掌握 fork, exec, waitpid 等函数的使用
问题描述：
父进程依次创建两个子进程，其中一个子进程运行"ls -l"指令，另一个进程在暂停 5s 后异常退出，父进程并不阻塞自己，并等待子进程的退出信息，待收集到该信息，父进程就返回。

5、多线程编程实验：利用线程参数重用一个线程函数创建多线程。
（1）创建两个新线程，一个输出 x，另一个输入 o，每个线程输出固定字符数后就从线程函数中返回退出线程。线程函数 char_print 在两个线程中均被执行，但是程序为每个线程指定不同的实例作为参数。
（2）引入 pthread_join 主线程收集两个子线程的退出信息，并释放资源。

6、互斥锁的使用方法　mutex_test.c
目的：掌握互斥锁来进行线程同步的方法。通过互斥锁来同步读、写线程对共享缓冲区的操作。

7、使用信号量同步线程实验　　sem_test1.c
目的：掌握使用信号量进行线程同步的方法。

(2) 修改程序使得三个线程的执行顺序为 a->b->c
sem_test2.c

8、条件变量同步线程实验
目的：掌握使用条件变量进行线程同步的方法。
cond_test.c

第四章 进程间的通信
1、kill+signal　实现不同进程发送信号机制　kill_signal_expr.c
创建一个子进程，父进程通过 kill 发送一个 SIGTERM 信号给子进程，子进程借助 signal 建立 SIGTERM 与信号处理函数的关联，当子进程接收到 SIGTERM 信号，使用 execl 函数终止自己。
提示：注意到 execl 函数的参数都是 char *类型，因此必须使用 sprintf 将 pid 从 int 转换成 char *,
eg:
char s[];
sprintf(s,"%d",getpid());

2、有名管道（命名管道）fifo_read.c, fifo_write.c
编写两个程序，一个用于读管道，另一个用于写管道。其中在读管道的程序里创建管道，并且作为 main 函数里的参数由用户输入要写入的内容。读管道读出用户写入管道的内容。
要求：这两个函数用的是非阻塞读写管道。
eg:
fd = open(FIFO, O_RDONLY | O_NONBLOCK, 0755);
fd = open(FIFO, O_WRONLY | O_NONBLOCK, 0);

3、信号量综合实例　ipc_sem_test.c
问题描述：首先实现两个函数，用于请求和释放信号量，每个进程只能请求一次，定义一个全局变量 semheld
来记录请求次数，当 semheld 大于 0 时不再增加信号量，当 semheld 小于 1 时，不再释放信号量。

4、信号量应用实例　sem_p_v.c
问题描述：此应用程序实现了父子进程间对信号量的 PV 操作。所谓 P 操作，就是将信号量的值减
去 1；所谓 V 操作，就是将信号量的值加 1。
在此程序中，子进程创建了一个含有一个信号量的信号量集合，并初始化为 5。在前 5 次 V 操作中，子进程每次 V 操作等待 1 秒，在后 5 次 V 操作中，子进程每次 V 操作等待 3
秒。父进程执行 10 次 P 操作，每次 P 操作等待 2 秒。

5、实现一个简单的消息队列工具，用于创建消息队列、发送、读取消息、改变权限以及删除消息队列。 msgtool 实例， msgtool.c
实现如下功能：

（1）　发送消息
msgtool s　(type) “text”
（2）　读取消息
msgtool r (type)
（3）　改变权限
msgtool m (mode)
（4）　删除队列
msgtool d

6、消息队列应用实例练习：　msg_receiver_example.c, msg_sender_example.c
原理及功能说明
此实例是一个简单的使用消息队列进行实时聊天的本机通信程序，发送端每发送一个消息，会立即被接收读取，在没有消息在消息队列中时，将处于阻塞状态。其运行结果如下：
（1）终端 1 运行接收端。
[root@localhost hanson]# ./msg_receiver_example //执行接收端
最开始执行时，在消息队列中没有信息，处于阻塞状态。
（2）终端 2 运行发送端。
[root@localhost hanson]# ./msg_sender_example //执行发送端
Enter the mssage to send:hello //输入信息
Enter the mssage to send:yes
Enter the mssage to send:go
Enter the mssage to send:end //结束信号
（3）有信息发送后，终端 1 接收到的信息。
receiver mssage:hello
receiver mssage:yes
receiver mssage:go
receiver mssage:end //通信结束
在整个过程中，可以通过 ipcs －q 命令查看，以下验证了接收端接收到消息后，该消息将从消息队列中删除。
[root@localhost hanson]# ipcs －q //使用 ipcs －q 查看当前消息队列情况
------ Message Queues --------
key msqid owner perms used-bytes messages //无任何消息队列
[root@localhost hanson]# ./msg_sender_example //执行发送消息命令
Enter the mssage to send:hello
Enter the mssage to send:world
Enter the mssage to send:embedded
Enter the mssage to send:end //结束发送消息
[root@localhost]# ipcs －q//查看当前消息队列使用情况

------ Message Queues --------
key msqid owner perms used-bytes messages

0x00003039 65536 root 666 2048 4 //包括详细的使用情况

[root@localhost yangzongde]# ./msg_receiver_example //从消息队列中接收消息
receiver mssage:hello
receiver mssage:world
receiver mssage:embedded
receiver mssage:end //接收完所有消息
[root@localhost yangzongde]# ipcs －q//再次查看
------ Message Queues --------
key msqid owner perms used-bytes messages //无任何消息内容

7、实现一个简单的共享内存工具 shmtool，用于创建共享内存、写读共享内存、改变权限以及删除共享内存。在读写操作的时候，如果共享内存已经存在，直接进行绑定并读写，否则先创建一个共享内存。
具体用法如下：
（1）把字符串写入共享内存
shmtool w "text"
（2）从共享内存中读取文件
shmtool r
（3）改变权限
shmtool m (mode)
（4）删除共享内存
shmtool d

共享内存工具 shmtool.c

8、共享内存处理应用示例 1　　shm_expr.c
共享内存实验
实验内容：简单实现在一个进程中利用共享内存实现对文件的打开、读写操作。（暂时不涉及共享内存同步机制，后续实验）
实验步骤：
（1）创建共享内存
（2）映射共享内存
（3）打开共享内存
（4）先写一个字符串“hello”至共享内存，然后将共享内存中字符串数据写入文件中，最后读出文件中字符串内容
（5）脱离共享内存绑定
（6）删除共享内存

9、共享内存处理应用示例 2　　parent_child_shm_example.c
功能描述 ：
此程序实现父子进程通过共享内存进行数据通信。子进程创建一个共享内存单元，然后子进程接受用户输入的信息（通过 argv[1]输入）， 并将其写入到共享内存单元；
父进程则等待直到子进程退出，再从共享内存单元将该信息读出，并显示信息的个数。此程序编译运行结果如下：

[root@localhost shmemory]# gcc -o parent_child_shm_example parent_child_shm_example.c //编译
[root@localhost shmemory]# ./parent_child_shm_example www.google.cn //运行
this is child.
write argv[1] to shm.
you input charater count is 13
this is parent.
input charater is www.google.cn//父进程显示写入到共享内存的信息

10、共享内存处理应用示例 3
shm_sem_example_send.c
shm_sem_example_recv.c
1．功能描述
此程序实现没有亲缘关系的两个进程间通过共享内存进行数据通信，同时，使用信号量来保证两个进程的读写同步：即发送方在发送数据时，接收方不能接收数据；接收方在接收数据时，发送方不能发送数据。
在代码使用中，使用共享内存来传递数据，使用信号量来同步读写端（此处仅使用二元信号量）。基本思路如下：
1)首先设置信号量初始值为 0，表示没有写入任何数据，不可以读。
2) 发送端在信号量的值为 0 时写入一段数据到共享内存中并阻塞读进程，写入完成后，置信号量的值为 1，表示可以读数据。此时也不能再写数据了。
3) 接收端在信号量的值为 1 时读出数据并阻塞写入端，读出完成后，设置信号量的值为 0，表示读出完成，可以再写数据。
发送端程序编译运行结果如下：
[root@localhost shared_memory]# gcc -o shm_sem_example_send shm_sem_example_send.c//编译
[root@localhost shared_memory]# ./shm_sem_example_send //运行发送端
write data operate //提示信号量值满足写操作，执行写操作，写时不能读数据
please input something:hello //要求输入数据，此处写入 hello
write data operate //提示信号量值满足写操作，执行写操作，写时不能读数据
please input something:world //要求输入数据，此处写入 world
write data operate //提示抢占到信号量，执行写操作，写时不能读数据
please input something:end //提示信号量值满足写操作，此处写入 end
接收端程序编译运行结果如下：
[root@localhost shared_memory]# gcc -o shm_sem_example_recv shm_sem_example_recv.c //编译
[root@localhost shared_memory]# ./shm_sem_example_recv //运行接收端
read data operate //提示信号量值满足读操作，执行读操作，读时不能写数据
hello
read data operate //提示信号量值满足读操作，执行读操作，读时不能写数据
world

在运行过程中，使用命令“ipcs”命令可以查看到使用的共享内存和信号量信息。
[root@localhost ~]# ipcs
------ Shared Memory Segments -------- //共享内存信息
key shmid owner perms bytes nattch status
0x0009fbf1 458752 root 600 2048 2

------ Semaphore Arrays -------- //信号量信息
key semid owner perms nsems
0x0001e240 458752 root 666 1
------ Message Queues --------
key msqid owner perms used-bytes messages

第五章 网络编程（上）
1、编程实现判断主机字节序
byte_order.c

2、编写域名解析程序
domain.c

3、基于 TCP 协议，实现一个简单的 ECHO 单客户端服务器通信程序。
ECHO 客户端接收用户输入发送给服务器端的 2029 端口，接收服务器端的响应并将它显示给用户。
这里，ECHO 服务器只能同时服务于一个客户端。

示例 5-3 单客户 echo 服务器
server.c, mynet.h, client.c

4、基于 TCP 实现一个多客户端并发 ECHO 服务器。
并发 ECHO 服务器同时可以服务于多个客户端，每当有客户端连接到来时，它通过调用 fork() 派送子进程为客户端提供服务。
示例 5-4　并发多客户 echo 服务器　　server.c, mynet.h, client.c

第六章 网络编程（下）
1、使用 select 实现 echo 多客户服务器程序。
示例 6-1 server.c, client.c

2、基于 UDP 协议，实现一个简单的 ECHO 单客户端服务器通信程序。
ECHO 客户端接收用户输入发送给服务器端的 2029 端口，接收服务器端的响应并将它显示给用户。
这里，ECHO 服务器只能同时服务于一个客户端。

示例 6-5 单客户 echo 服务器

server.c, mynet.h, client.c

3、使用广播的 echo 单客户端服务器通信程序　　示例：6-8
客户端用户输入受限广播地址如(255.255.255.255)，也可以输入子网的广播地址（eg: 192.168.1.255），在收到 echo 服务器的应答时，将输出服务器的地址，以确定是谁发送了应答。
测试如下：
运行服务器端程序：　#./server
运行客户端程序：./client 192.168.1.255 或者./client 255.255.255.255
Enter the message:　hello
Relay from: 192.168.1.72 2029
Server echo　message: hello

4、示例 6-10：ECHO 守护进程(服务器端)
修改多客户端服务器程序 ECHO，调用 make_daemon()之后使之变成守护进程。

5、基于 xinetd 守护进程创建 echo 服务器　　示例：6-11