**《Linux操作系统设计实践》**

**实验报告**

学号：031602504

姓名：陈俊杰

年级：2016级

学院：数学与计算机科学学院

专业：网络工程

实验时间：2018－2019学年第一学期

目录

[**实验一：进程管理** 3](#_Toc528966048)

[**实验环境：** 3](#_Toc528966049)

[**实验内容：** 3](#_Toc528966050)

[**实验总结：** 7](#_Toc528966051)

[**实验二：进程通信** 9](#_Toc528966052)

[**实验环境：** 9](#_Toc528966053)

[**实验内容：** 9](#_Toc528966054)

[**实验总结：** 22](#_Toc528966055)

[**实验三：网络通信** 23](#_Toc528966056)

[**实验环境：** 23](#_Toc528966057)

[**实验内容：** 23](#_Toc528966058)

[**实验总结：** 36](#_Toc528966059)

[**实验四：文件操作** 37](#_Toc528966060)

[**实验环境：** 37](#_Toc528966061)

[**实验内容：** 37](#_Toc528966062)

[**实验总结：** 45](#_Toc528966063)

[**实验五：图形界面** 47](#_Toc528966064)

[**实验环境：** 47](#_Toc528966065)

[**实验内容：** 47](#_Toc528966066)

[**实验总结：** 57](#_Toc528966067)

[**实验六：综合应用** 60](#_Toc528966068)

[**实验环境：** 60](#_Toc528966069)

[**实验内容：** 60](#_Toc528966070)

[**实验总结：** 72](#_Toc528966071)

**学号：031602504 姓名：陈俊杰 学院：数计学院**

**专业：网络工程 班级：计算机5班**

**《Linux 操作系统设计实践》**

**实验一：进程管理**

**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

**1.关于C 语言编程方法，Linux 下编译、调试、运行C 程序的方法：**

**在ubunt里直接使用gedit（类似写字板）或者使用vim进行C语言代码编写，然后在控制台使用命令编译**

**cc –o example.exe example.c**

**之后生成一个example.exe的可执行文件**

**接着在控制台命用命令**

**./example.out**

**运行编译过的程序**

**2. fork ( ) 的作用，进程的创建过程：**

**这里，我写的代码**

#include <stdio.h>

int main()

{

int i;

if ( fork() == 0 ) /\*创建新的子进程,并判断是否是子进程\*/

{

/\* 子进程程序 \*/

for ( i = 0; i < 5; i++ )

{

printf("This is child process %d \n",i);

sleep(1);

}

}

else

{

/\* 父进程程序\*/

for ( i = 0; i < 5; i++ )

{

printf("This is parent process %d \n",i);

sleep(1);

}

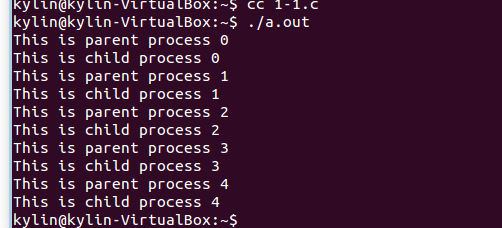
}

}

/\*运行时屏幕上交替出现子进程与父进程各打印出的5条信息\*/

/\*注意观察出现的交替情况每次可能会不一样,注意观察命令提示符出现的位置\*/

**如：**

****

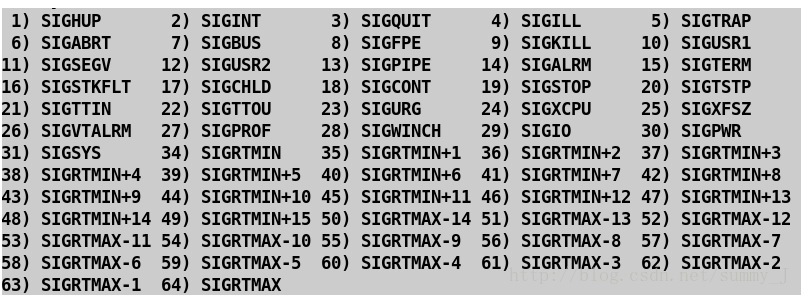
**3.尝试加入信号使得进程之间可以相互影响**

**signal（参数1，参数2）；**

**参数1：我们要进行处理的信号。系统的信号我们可以再终端键入 kill -l查看(共64个)。其实这些信号时系统定义的宏。**

**参数2：我们处理的方式（是系统默认还是忽略还是捕获）。**

**比如用**kill -l**命令查看系统中定义的信号列表**

****

**在命令行上输man n（对应想查的序号的信号的作用） signal，就可以看相应的功能比如**

（1）SIG\_IGN

这个符号表示忽略该信号。

#include <stdio.h>

#include <signal.h>

int main(int argc, char \*argv[]) {

signal(SIGINT, SIG\_IGN);

while(1);

return 0;

}

SIGINT信号代表由InterruptKey产生，通常是CTRL +C 或者是DELETE 。执行上述代码时，按下CTRL + C程序没有反应。这就对了，如果我们想结束该程序可以按下CTRL +\来结束，当我们按下CTRL +\组合键时，产生了SIGQUIT信号，此信号并没有被忽略。

（2）SIG\_DFL

这个符号表示恢复对信号的系统默认处理。不写此处理函数默认也是执行系统默认操作

#include <stdio.h>

#include <signal.h>

int main(int argc, char \*argv[]) {

signal(SIGINT, SIG\_DFL);

while(1);

return 0;

}这时就可以按下CTRL +C 来终止该进程了。把signal(SIGINT, SIG\_DFL);这句去掉，效果是一样的

**然后在以上程序中加入简单信号**

**#include <stdio.h>**

**#include<signal.h>**

**int main()**

**{**

**int i;**

**if ( fork() == 0 ) /\*创建新的子进程,并判断是否是子进程\*/**

**{**

**/\* 子进程程序 \*/**

**for ( i = 0; i < 5; i++ )**

**{**

**printf("This is child process %d \n",i);**

**sleep(1);**

**}**

**while(1);**

**}**

**else**

**{**

**signal(SIGINT,SIG\_IGN);**

**printf("child process will be killed by ctrl+C,but father not");**

**/\* 父进程程序\*/**

**for ( i = 0; i < 5; i++ )**

**{**

**printf("This is parent process %d \n",i);**

**sleep(1);**

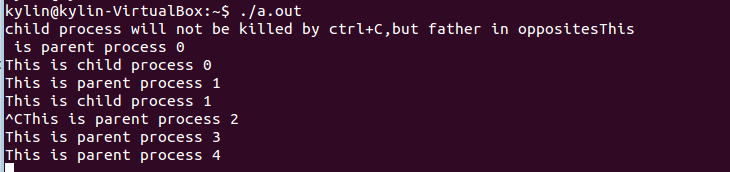
**}**

**while(1);**

**}**

**}**

**结果则是：**

****

**子进程被ctrl+c kill 掉，而父进程将信号忽略。**

**再改：在父进程中再加个signal(SIGINT,SIG\_DFL);**

**于是结果是进程都被停掉了。**

**实验总结：**

**1．首先是进程和程序的概念：进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是**[**操作系统**](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)**结构的基础。而程序是一个指令序列。**

**2.然后是并发执行的实质：一组在逻辑上互相独立的**[**程序**](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F/13831935)**或程序段在执行过程中，其执行时间在客观上互相重叠，即一个程序段的执行尚未结束，另一个程序段的执行已经开始的这种执行方式。**

**3.** **线程概念：在一个程序里的一个执行路线就叫做线程。更准确的定义是：线程是“一个进程内部的控制序列”。一切进程至少都有一个执行线程**

**4.** **信号概念：在计算机科学中，信号是Unix、类Unix以及其他POSIX兼容的操作系统中进程间通讯的一种有限制的方式。它是一种异步的通知机制，用来提醒进程一个事件已经发生。当一个信号发送给一个进程，操作系统中断了进程正常的控制流程，此时，任何非原子操作都将被中断。如果进程定义了信号的处理函数，那么它将被执行，否则就执行默认的处理函数。**

**实验过程中对以上概念的了解更加深刻，由于对其的多次应用以及对其的思考。在理解进程的过程中写了关于子进程与父进程之间的纠葛，对进程并发过程有更形象的理解。从父子进程的数据变化中了解到，一旦子进程创建成功后它们数据就分开了，不共享任何数据了，为子进程准备了新的代码段，数据段堆栈段。还有在时间片足够程序运行的情况下父子进程交叉的概率是很小的，因此在代码中引入sleep更易于观察。对信号的理解在对程序的修改中得到了更好的理解，对其可以进行较简单的应用了。比如忽略信号ctrl+c的操作，恢复默认信号的操作。**

**学号：031602504姓名：陈俊杰 学院：数学与计算机科学学院**

**专业：网络工程 班级：计算机5班**

**《Linux 操作系统设计实践》**

**实验二：进程通信**

**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

1. **Linux下IPC(Inter-Process Communication，进程间通信)机制：**

**半双工管道、命名管道、消息队列、信号、信号量、共享内存、内存映射文件，套接字等等，这里介绍信号量和共享内存机制**

1. **信号量：**
2. **定义：信号量(Semaphore)，有时被称为信号灯，是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被**[**并发**](https://baike.baidu.com/item/%E5%B9%B6%E5%8F%91/11024806)**调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量**
3. **专有的操作函数：**

**int  semget(key\_t  \_key ,int  \_nsems,int \_semflg);**

**功能：创建一个新的信号量或获取一个已经存在的信号量的键值。**

**返回值：成功返回信号量的标识码ID。失败返回-1；**

**参数1：\_key  为整型值，用户可以自己设定。有两种情况：键值是IPC\_PRIVATE，该值通常为0，意思就是创建一个仅能被进程进程给我的信号量。 键值不是IPC\_PRIVATE，我们可以指定键值，例如1234；也可以一个ftok()函数来取得一个唯一的键值。**

**参数2：\_nsems 表示初始化信号量的个数。比如我们要创建一个信号量，则该值为1.,创建2个就是2。**

**参数3：\_semflg  ：信号量的创建方式或权限。有IPC\_CREAT，IPC\_EXCL。**

**IPC\_CREAT如果信号量不存在，则创建一个信号量，否则获取。**

**IPC\_EXCL只有信号量不存在的时候，新的信号量才建立，否则就产生错误**

**int  semctl(int \_semid  ,int \_semnum,int \_cmd  ……);**

**功能：控制信号量的信息。**

**返回值：成功返回0，失败返回-1；**

**参数1：\_semid   信号量的标志码(ID)，也就是semget（）函数的返回值;**

**参数2：\_semnum,  操作信号在信号集中的编号。从0开始。**

**参数3：\_cmd    命令，表示要进行的操作。**

**参数\_cmd中可以使用的命令如下：**

**IPC\_STAT读取一个信号量集的数据结构semid\_ds，并将其存储在semun中的buf参数中。**

**IPC\_SET设置信号量集的数据结构semid\_ds中的元素ipc\_perm，其值取自semun中的buf参数。**

**IPC\_RMID将信号量集从内存中删除。**

**GETALL用于读取信号量集中的所有信号量的值。**

**GETNCNT返回正在等待资源的进程数目。**

**GETPID返回最后一个执行semop操作的进程的PID。**

**GETVAL返回信号量集中的一个单个的信号量的值。**

**GETZCNT返回这在等待完全空闲的资源的进程数目。**

**SETALL设置信号量集中的所有的信号量的值。**

**SETVAL设置信号量集中的一个单独的信号量的值。**

**int    semop(int   semid ,struct    sembuf   \*\_sops ,size\_t  \_nsops);**

**功能：用户改变信号量的值。也就是使用资源还是释放资源使用权。**

**返回值：成功返回0，失败返回-1；**

**参数1：\_semid : 信号量的标识码。也就是semget（）的返回值。**

**参数2： \_sops是一个指向结构体数组的指针。**

**struct   sembuf{**

**unsigned short  sem\_num;//第几个信号量，第一个信号量为0；**

**short  sem\_op;//对该信号量的操作。**

**short \_semflg;**

**};**

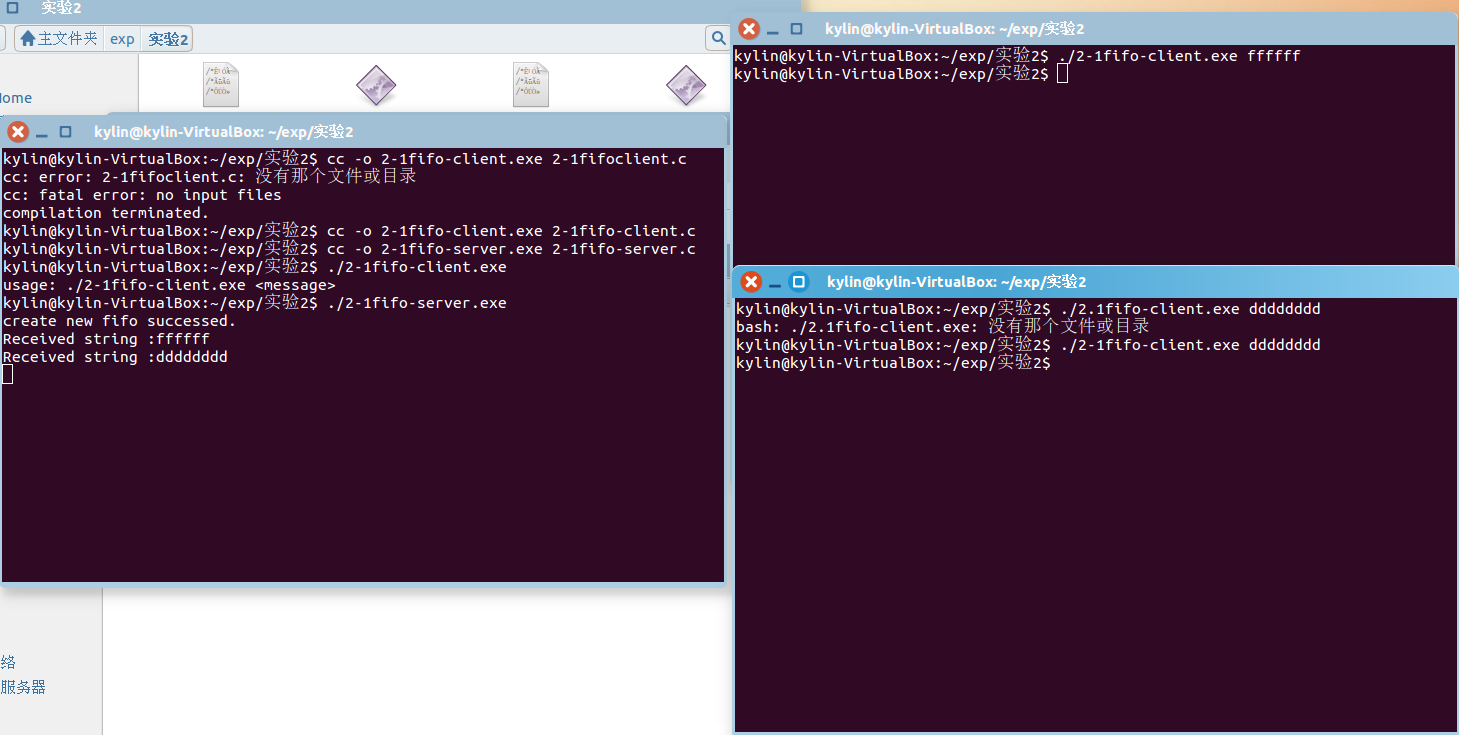
**sem\_num:  操作信号在信号集中的编号。第一个信号的编号为0；sem\_op : 如果其值为正数，该值会加到现有的信号内含值中。通常用于释放所控资源的使用权；如果sem\_op的值为负数，而其绝对值又大于信号的现值，操作将会阻塞，直到信号值大于或等于sem\_op的绝对值。通常用于获取资源的使用权；如果sem\_op的值为0，则操作将暂时阻塞，直到信号的值变为0。\_semflg 有IPC\_NOWAIT 对信号的操作不能满足时，semop()不会阻塞，并立即返回，同时设定错误信息。IPC\_UNDO 程序结束时(不论正常或不正常)，保证信号值会被重设为semop()调用前的值。这样做的目的在于避免程序在异常情况下结束时未将锁定的资源解锁，造成该资源永远锁定。**

1. **共享内存：**
2. **定义：共享内存指 (shared memory)在多处理器的计算机系统中，可以被不同中央处理器（CPU）访问的大容量内存。由于多个CPU需要快速访问存储器，这样就要对存储器器进行缓存存（Cache）。任何一个缓存的数据被更新后，由于其他处理器也可能要存取，共享内存就需要立即更新，否则不同的处理器可能用到不同的数据。共享内存是 Unix下的多进程之间的通信方法 ,这种方法通常用于一个程序的多进程间通信，实际上多个程序间也可以通过共享内存来传递信息。**
3. **相关函数**



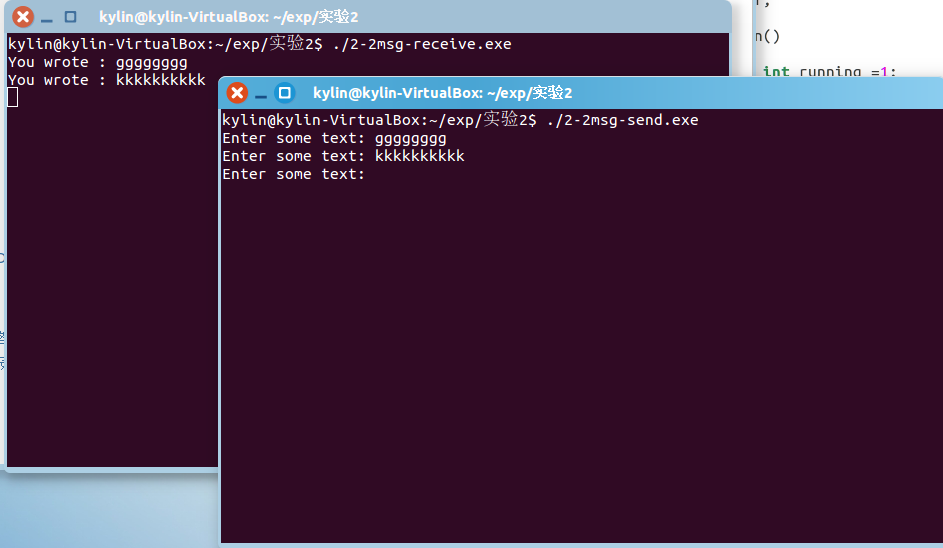
**这样资料就收集好了，然后分析示例程序代码：**

**2-1 fifo所涉及的不懂的函数：**

1. **umask设置了用户创建文件的默认权限，它与chmod的效果刚好相反，umask设置的是权限“补码”，而chmod设置的是文件权限码。设置为0，它的意思就是0取反再创建文件时权限相与，也就是： 0777；**
2. **mknod命令用于创建一个设备文件，即特殊文件，该函数最主要的用途就是创建FIFO文件，mknod(FIFO\_FILE，S\_IFIFO|0666,0);FIFO\_FILE是一个字符指针，指向文件名，S\_IFIFO表示要创建一个FIFO文件，0666表示该文件的权限是所有人可读可写，0表示该文件不是一个设备文件。运行结果**

**其服务器端创建一个FIFO文件，其余客户端向FIFO文件写入信息，然后服务端读出。**

**2-2运行结果：**

****

**系统调用消息队列相关函数对某个key值的消息队列进行读写，发送端写入，接收端接受消息**

**2-3类似的，创建一个共享内存空间，一个写，一个读。**

**于是用信号量与共享内存的资料写基于信号量与共享内存的模拟信箱之间的通信，先是对信号量函数进行一些相关的封装使其更好的调用比如：**

**union semun**

**{**

**int val;**

**struct semid\_ds\* buf;**

**unsigned short \* array;**

**struct seminfo\* \_buf;**

**};**

**struct sembuf sembufinit(int a,int b)**

**{**

**struct sembuf sembuft;**

**sembuft.sem\_num = a;**

**sembuft.sem\_op = b;**

**sembuft.sem\_flg = SEM\_UNDO;**

**return sembuft;**

**}**

**struct sembuf sembuf\_p=sembufinit(0,-1);**

**struct sembuf sembuf\_v=sembufinit(0,1);**

**以便于调用信号的pv操作，**

**总的来说代码如下**

**Mail.h**

**#include<sys/sem.h>**

**#include<sys/ipc.h>**

**#include<sys/shm.h>**

**#include<string.h>**

**#include<stdio.h>**

**#include<stdlib.h>**

**/\***

**struct sembuf**

**{**

**unsigned short int sem\_num; //xin hao liang de xuhao**

**short int sem\_op; // dui xin hao liang de cao zuo**

**short int sem\_flg; // cao zuo biao shi (qian xian)**

**}; \*/**

**union semun**

**{**

**int val;**

**struct semid\_ds\* buf;**

**unsigned short \* array;**

**struct seminfo\* \_buf;**

**};**

**struct sembuf sembufinit(int a,int b)**

**{**

**struct sembuf sembuft;**

**sembuft.sem\_num = a;**

**sembuft.sem\_op = b;**

**sembuft.sem\_flg = SEM\_UNDO;**

**return sembuft;**

**}**

**struct sembuf sembuf\_p=sembufinit(0,-1);**

**struct sembuf sembuf\_v=sembufinit(0,1);**

**struct date**

**{**

**char dates[100];**

**};**

**struct Mailbox**

**{**

**char boxname[100]; //信箱名**

**int bid; //标识符**

**int size; //可接受消息的数目**

**int mailnum,freenum,rmutex,wmutex,in,out; //一堆信号量**

**date buf[100]; //接收的消息**

**};**

**int Mailboxinit(Mailbox\* &a,char n[100],int s)/\*初始化的指针和信箱名和信箱长度\*/**

**{**

**int shmid = shmget(0,sizeof(struct Mailbox),IPC\_CREAT|0777); //创建一个共享内存对象并返回标识符**

**a = (Mailbox\*)shmat(shmid,0,0777); //映射**

**a->size = s;**

**strcpy(a->boxname,n);**

**a->mailnum= semget(0,1,IPC\_CREAT|0777); //创建信号量**

**a->freenum= semget(0,1,IPC\_CREAT|0777);**

**a->rmutex = semget(0,1,IPC\_CREAT|0777);**

**a->wmutex = semget(0,1,IPC\_CREAT|0777);**

**semctl(a->mailnum,0,SETVAL,0); //设置信号量的值**

**semctl(a->freenum,0,SETVAL,a->size);**

**semctl(a->rmutex,0,SETVAL,1);**

**semctl(a->wmutex,0,SETVAL,1);**

**a->in = a->out = 0;**

**a->bid = shmid;**

**return shmid;**

**}**

**void send(Mailbox\* &dest,date &mptr)**

**{**

**semop(dest->freenum,&sembuf\_p,1); //p操作**

**semop(dest->wmutex,&sembuf\_p,1);**

**strcpy(dest->buf[dest->in].dates,mptr.dates); //对消息进行操作**

**dest->in=(dest->in+1)%dest->size;**

**semop(dest->wmutex,&sembuf\_v,1); //v操作**

**semop(dest->mailnum,&sembuf\_v,1);**

**}**

**void receive(Mailbox\* &addr,date &mptr)**

**{**

**semop(addr->mailnum,&sembuf\_p,1);**

**semop(addr->rmutex,&sembuf\_p,1);**

**strcpy(mptr.dates,addr->buf[addr->out].dates);**

**addr->out = (addr->out+1)%addr->size;**

**semop(addr->rmutex,&sembuf\_v,1);**

**semop(addr->freenum,&sembuf\_v,1);**

**}**

**void delshm(Mailbox\* &addr,int id1)**

**{**

**shmdt((void\*)addr);**

**shmctl(id1,IPC\_RMID,0);**

**}**

**Mail.c**

**#include"Mail.h"**

**int main()**

**{**

**Mailbox \*a,\*b;**

**int thismailid,thatmailid;**

**char name[100];**

**struct date mptr;**

**int ismailed = 0,mailsize,items=0;**

**char add[] = " from 邮箱";**

**while(1)**

**{**

**printf("信箱未创建,是否创建？\nyes:1\tno:0\texit:2\n");**

**scanf("%d",&ismailed);**

**if(ismailed==1)**

**{**

**printf("请输入创建信箱名\n");**

**scanf("%s",name);**

**printf("请输入创建信箱大小（可接收的信件数量）：\n");**

**scanf("%d",&mailsize);**

**thismailid = Mailboxinit(a,name,mailsize);**

**printf("创建成功!\n");**

**while(1)**

**{**

**printf("你的信箱id为%d\n已接受数量/可接收数量 : %d/%d\n",a->bid,a->in-a->out,a->size);**

**printf("接收:1 发送:2 撤销信箱:3 刷新:其他\n");**

**scanf("%d",&items);**

**if(items==1)**

**{**

**receive(a,mptr);**

**printf("接收的信息为:\n%s\n\n",mptr.dates);**

**}**

**else if(items==2)**

**{**

**while(1){**

**printf("选择发送对象的邮箱id\n");**

**scanf("%d",&thatmailid);**

**b =(struct Mailbox\*)shmat(thatmailid,0,0777);**

**if(b) break;**

**printf("此邮箱id不可用!\n");**

**}**

**printf("请输入要发送的内容:\n");**

**scanf("%s",mptr.dates);**

**strcat(mptr.dates,add);**

**strcat(mptr.dates,a->boxname);**

**send(b,mptr);**

**printf("发送成功!");**

**}**

**else if(items==3)**

**{**

**delshm(a,thismailid);**

**break;**

**}**

**else**

**{**

**;**

**}**

**}**

**}**

**else if(ismailed==2)**

**break;**

**else continue;**

**}**

**return 0;**

**}**

**运行结果：**

****

**实验总结：**

**实验使我了解到Linux下IPC(Inter-Process Communication，进程间通信)机制：半双工管道、命名管道、消息队列、信号、信号量、共享内存、内存映射文件，套接字等等，这里介绍信号量和共享内存机制这些的概念由于去百度等各种途径了解与学习，然后就跑四个例子了解其代码函数的含义因而又有收获，在学习思考了解给的例子的基础上结合上学期计算机系统的学习与有次类似的作业即用共享内存与信号量机制实现信箱通信，在利用上学期学过的知识与作业的立场下对其进行回忆并进行修改总结后写了这个表现出对共享内存和信号量的掌握的程序，其过程还是比较顺利的，毕竟是有了解学习过，总的来说，这次实验又一次的应用对其的有更好的感觉。**

**学号：031602504姓名：陈俊杰 学院：数学与计算机科学学院**

**专业：网络工程 班级：计算机5班**

**《Linux 操作系统设计实践》**

**实验三：网络通信**

**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

1. **套接口：通讯的基石是套接口，一个套接口是通讯的一端。在这一端上你可以找到与其对应的一个名字。一个正在被使用的套接口都有它的类型和与其相关的进程。套接口存在于通讯域中。通讯域是通过套接口通讯来处理一般的线程而引进的一种**[**抽象**](https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1/82058)**概念。**
2. **分析示例：**

**3-1>**

**sockaddr\_in（在netinet/in.h中定义）：**

**struct sockaddr\_in**

**{**

**short sin\_family;/\*Address family一般来说AF\_INET（地址族）PF\_INET（协议族）\*/**

**unsigned short sin\_port;/\*Port number(必须要采用网络数据格式,普通数字可以用htons()函数转换成网络数据格式的数字)\*/**

**struct in\_addr sin\_addr;/\*IP address in network byte order（Internet address）struct in\_addr { in\_addr\_t s\_addr;}结构体in\_addr 用来表示一个32位的IPv4地址.\*/**

**unsigned char sin\_zero[8];/\*Same size as struct sockaddr没有实际意义,只是为了　跟SOCKADDR结构在内存中对齐\*/**

**}；**

**stderr -- 标准错误输出设备**

**char\*strerror(int errnum);** **errnum：错误标号，通常用errno(标准错误号，定义在errno.h中)** **返回：指向错误信息的指针（即：错误的描述字符串）**

**extern void bzero（void \*s, int n）;参数说明：s 要置零的数据的起始地址； n 要置零的数据字节个数。在POSIX.1-2008标准里已经没有这些函数了。推荐使用memset替代bzero**

**uint32\_t htonl(uint32\_t hostlong); 将主机数转换成无符号长整形的网络字节顺序。#include <arpa/inet.h>**

**INADDR\_ANY：转换过来就是0.0.0.0，泛指本机的意思，也就是表示本机的所有IP，因为有些机子不止一块网卡，多网卡的情况下，这个就表示所有网卡ip地址的意思。**

**int socket( int af, int type, int protocol); 创建一个套接口（）。#include <sys/socket.h>af：一个地址描述。目前仅支持AF\_INET格式，也就是说ARPA Internet地址格式。type：指定socket类型。新套接口的类型描述类型，如TCP（SOCK\_STREAM）和UDP（SOCK\_DGRAM）。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等。protocol：顾名思义，就是指定协议。套接口所用的协议。如调用者不想指定，可用0。常用的协议有，IPPROTO\_TCP、IPPROTO\_UDP、IPPROTO\_STCP、IPPROTO\_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议。若无错误发生，socket()返回引用新套接口的描述字。否则的话，返回INVALID\_SOCKET错误，应用程序可通过WSAGetLastError()获取相应错误代码。**

**struct sockaddr**

**{**

**unsigned short sa\_family;/\*addressfamily,AF\_xxx\*/**

**char sa\_data[14];/\*14bytesofprotocoladdress\*/**

**};sa\_family是地址家族，一般都是“AF\_xxx”的形式。通常大多用的是都是AF\_INET,代表TCP/IP协议族。sa\_data是14字节协议地址。**

**bind()将一本地地址与一套接口捆绑。本函数适用于未连接的数据报或流类套接口，在connect()或listen()调用前使用。当用socket()创建套接口后，它便存在于一个名字空间（地址族）中，但并未赋名。bind()函数通过给一个未命名套接口分配一个本地名字来为套接口建立本地捆绑（主机地址/端口号）。**

**int listen(int sockfd, int backlog);#include <sys/socket.h>sockfd：用于标识一个已捆绑未连接套接口的描述字。backlog：等待连接队列的最大长度。如无错误发生，listen()返回0。否则的话，返回-1**

**SOCKET accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);参数sockfd：套接字描述符，该套接口在listen()后监听连接。addr：（可选）指针，指向一缓冲区，其中接收为通讯层所知的连接实体的地址。Addr参数的实际格式由套接口创建时所产生的地址族确定。**

**addrlen：（可选）指针，输入参数，配合addr一起使用，指向存有addr地址长度的整型数。如果没有错误产生，则accept()返回一个描述所接受包的SOCKET类型的值。否则的话，返回INVALID\_SOCKET错误**

**gethostbyname()返回对应于给定主机名的包含主机名字和地址信息的hostent结构的指针**

**struct hostent**

**{ char \*h\_name;**

**char \*\* h\_aliases;**

**short h\_addrtype;**

**short h\_length;**

**char \*\* h\_addr\_list;};**

**connect()用于建立与指定socket的连接**

**int connect(SOCKET s, const struct sockaddr \* name, int namelen);**

**#include <sys/socket.h>**

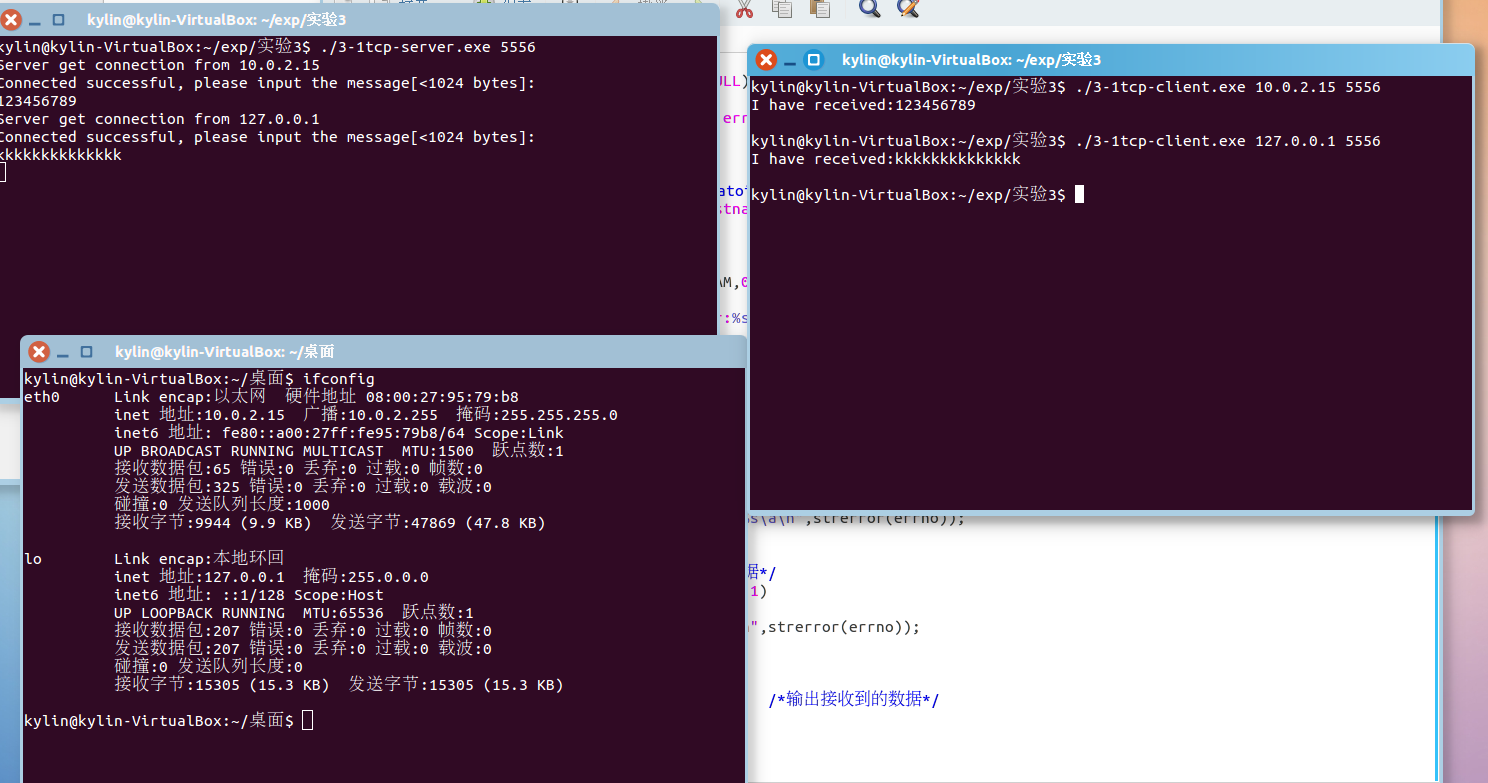
**参数:s：标识一个未连接**

**socket name：指向要连接套接字的sockaddr结构体的**

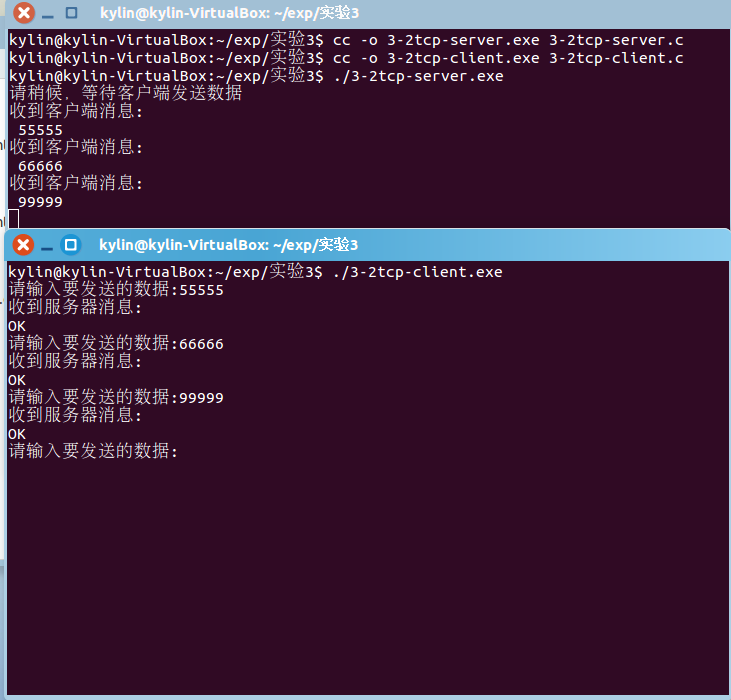
**namelen：sockaddr结构体的字节长度**

**ssize\_t read (int fd, void \*buf, size\_t count);read()会把参数fd所指的文件传送nbyte个字节到buf指针所指的内存中。若参数nbyte为0，则read()不会有作用并返回0。返回值为实际读取到的字节数，如果返回0，表示已到达文件尾或无可读取的数据。错误返回-1,并将根据不同的错误原因适当的设置错误码。**

**结果**

****

**3-2>:**

****

**解读了示例代码后对套接口编程有了一定了解，于是在解读了示例的基础上尝试自己写一个类似的：**

**源代码：**

**tcp-server.c**

**#include<stdio.h>**

**#include<netinet/in.h>**

**#include<sys/socket.h>**

**#include<string.h>**

**#include<unistd.h>**

**#include <netdb.h>**

**#include<stdlib.h>**

**#include <arpa/inet.h>**

**#define PORT 3333**

**int main(int argc,char \* argv[])**

**{**

**int sockfd,new\_fd=0;**

**int sinlen,wrchoose=0;**

**struct sockaddr\_in server\_addr; /\*定义服务器端套接口数据结构server\_addr \*/**

**struct sockaddr\_in client\_addr; /\*定义客户端套接口数据结构client\_addr \*/**

**char buf[1000];**

**sockfd = socket(AF\_INET,SOCK\_STREAM,0); /\* 创建套接字\*/**

**memset(&server\_addr,'0',sizeof(struct sockaddr\_in)); /\*将套接口地址数据结构清零\*/**

**server\_addr.sin\_family=AF\_INET; /\*设为TCP/IP地址族\*/**

**server\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY); /\*设为本机地址\*/**

**server\_addr.sin\_port=htons(PORT);**

**server\_addr.sin\_port = PORT; /\* 定义服务器套接字端口\*/**

**bind(sockfd,(struct sockaddr \*)(&server\_addr),sizeof(struct sockaddr)); /\*调用bind函数绑定端口\*/**

**listen(sockfd,1);**

**while(1)**

**{**

**if(!new\_fd)**

**{**

**sinlen = sizeof(struct sockaddr\_in);**

**new\_fd=accept(sockfd,(struct sockaddr \*)(&client\_addr),&sinlen);**

**printf("Server get connection from %s\n",inet\_ntoa(client\_addr.sin\_addr));**

**/\*TCP连接已建立，打印申请连接的客户机的IP地址\*/**

**}**

**printf("write or read ? \nwrite:1\tread:2\n");**

**scanf("%d",&wrchoose);**

**if(wrchoose==1)**

**{**

**printf("please input the massage[<1000 bytes]:\n");**

**scanf("%s",buf);**

**send(new\_fd,buf,strlen(buf),0);**

**printf("send success\n");**

**wrchoose=0;**

**}**

**if(wrchoose==2)**

**{**

**if(recv(new\_fd,buf,1000,0)>0)**

**{**

**printf("received :%s\n",buf);**

**wrchoose=0;**

**}**

**}**

**}**

**close(new\_fd);**

**close(sockfd);**

**return 0;**

**}**

**tcp-client.c**

**#include<stdio.h>**

**#include<netinet/in.h>**

**#include<sys/socket.h>**

**#include<string.h>**

**#include<unistd.h>**

**#include <netdb.h>**

**#include<stdlib.h>**

**#include <arpa/inet.h>**

**#define PORT 3333**

**int main(int argc,char\* argv[])**

**{**

**int sockfd;**

**int wrchoose=0;**

**struct hostent \*host;**

**if(argc!=2)**

**{ /\*获得命令行的第二个参数-主机名\*/**

**printf("Usage:%s hostname \a\n",argv[0]);**

**exit(1);**

**}**

**if((host=gethostbyname(argv[1]))==NULL)**

**{ /\*获得命令行的第二个参数-主机名\*/**

**fprintf(stderr,"Gethostname error\n");**

**exit(1);**

**}**

**struct sockaddr\_in server\_addr; /\*定义服务器端套接口数据结构server\_addr \*/**

**char buf[1000];**

**sockfd = socket(AF\_INET,SOCK\_STREAM,0); /\* 创建套接字\*/**

**memset(&server\_addr,'0',sizeof(struct sockaddr\_in)); /\*将套接口地址数据结构清零\*/**

**server\_addr.sin\_family=AF\_INET; /\*设为TCP/IP地址族\*/**

**server\_addr.sin\_addr=\*((struct in\_addr \*)host->h\_addr);**

**server\_addr.sin\_port=htons(PORT);**

**server\_addr.sin\_port = PORT; /\* 定义服务器套接字端口\*/**

**connect(sockfd,(struct sockaddr \*)(&server\_addr),sizeof(struct sockaddr));**

**while(1)**

**{**

**printf("get connection to %s\n",inet\_ntoa(server\_addr.sin\_addr));**

**printf("write or read ? \nwrite:1\tread:2\n");**

**scanf("%d",&wrchoose);**

**if(wrchoose==1)**

**{**

**printf("please input the massage[<1000 bytes]:\n");**

**scanf("%s",buf);**

**send(sockfd,buf,strlen(buf),0);**

**printf("send success\n");**

**wrchoose=0;**

**}**

**if(wrchoose==2)**

**{**

**if(recv(sockfd,buf,1000,0)>0)**

**{**

**printf("received :%s\n",buf);**

**wrchoose=0;**

**}**

**}**

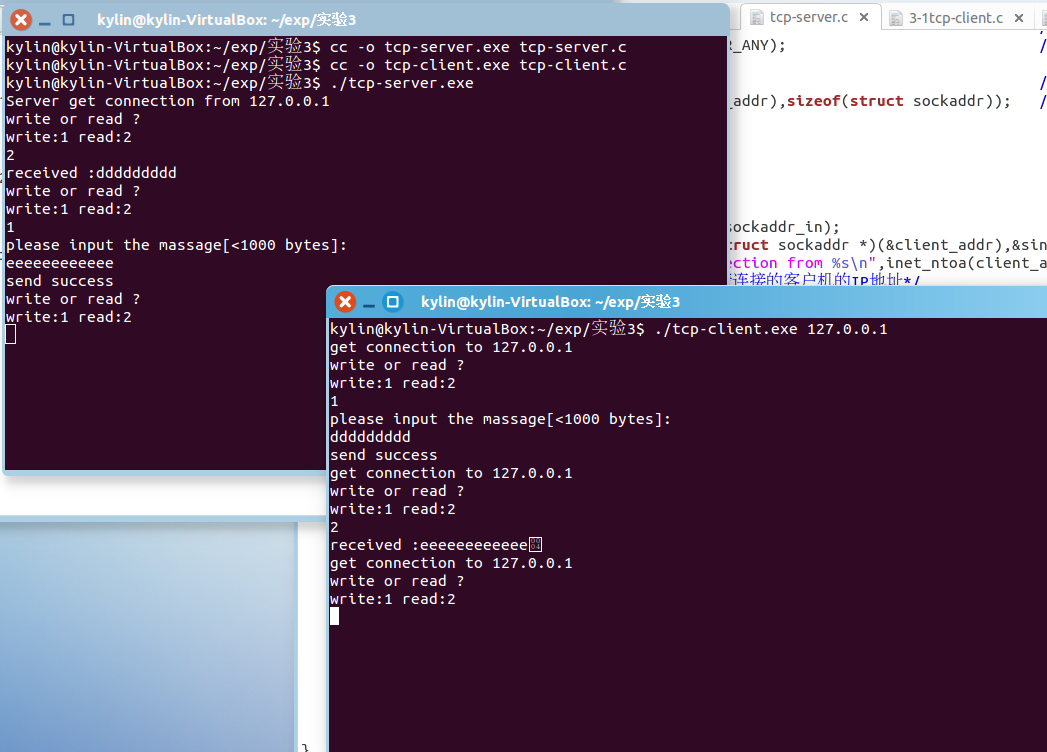
**}**

**close(sockfd);**

**return 0;**

**}**

**在两个示例的基础上稍加修改，调试结果：**

****

**服务端客户端可互发简单消息**

**实验总结：**

**收获还是很多的，从示例中不懂的各种百度，资料看了好久，从开始的不理解懵的情况下一个一个结构体函数用法百度过去，在理解代码的过程中对套接字网络编程有了一定的了解，学习成果颇为丰富，对以前学过的知识进行了实践然后反馈给对理论的更好理解，对其整个过程就是服务端创建监听接受连接接收消息发送消息，客户端连接接收发送等过程。在写代码的时候倒是没遇到特别的问题，毕竟是模仿两个示例，总的来说是大致了解掌握了使用套接口通信机制实现客户端进程与服务器端进程之间信息的 发送接收。**

**学号：031602504姓名：陈俊杰 学院：数学与计算机科学学院**

**专业：网络工程 班级：计算机5班**

**《Linux 操作系统设计实践》**

**实验四：文件操作**

**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

**fopen（）**

**函数原型 函数原型：FILE \* fopen(const char \* path, const char \* mode);**

**FILE \*fp ;**

**fp = fopen("D:\\a.txt","r");**

**\\是一种转义字符，他表示一个\，就像\n表示回车一样，即可用绝对路径也可用相对路径**

**mode：**

| **字符串** | **说明** |
| --- | --- |
| **r** | **以只读方式打开文件，该文件必须存在。** |
| **r+** | **以读/写方式打开文件，该文件必须存在。** |
| **rb+** | **以读/写方式打开一个二进制文件，只允许读/写数据。** |
| **rt+** | **以读/写方式打开一个文本文件，允许读和写。** |
| **w** | **打开只写文件，若文件存在则长度清为 0，即该文件内容消失，若不存在则创建该文件。** |
| **w+** | **打开可读/写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。** |
| **a** | **以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留（EOF 符保留）。** |
| **a+** | **以附加方式打开可读/写的文件。若文件不存在，则会建立该文件，如果文件存在，则写入的数据会被加到文件尾后，即文件原先的内容会被保留（原来的 EOF 符不保留）。** |
| **wb** | **以只写方式打开或新建一个二进制文件，只允许写数据。** |
| **wb+** | **以读/写方式打开或建立一个二进制文件，允许读和写。** |
| **wt+** | **以读/写方式打开或建立一个文本文件，允许读写。** |
| **at+** | **以读/写方式打开一个文本文件，允许读或在文本末追加数据。** |
| **ab+** | **以读/写方式打开一个二进制文件，允许读或在文件末追加数据。** |

**fopen\_s（）**

**函数原型：  
errno\_t fopen\_s( FILE\*\* pFile, const char *filename, const char mode );  
errno\_t \_wfopen\_s( FILE* pFile, const wchar\_t \*filename, const wchar\_t \*mode );**

**errno\_t是一种数据类型，实际上是一个整形，代表错误号码，如果是0就代表没有错误，其他代表抛出一个错误。**

**char是8位字符类型，最多只能包含256种字符，许多外文字符集所含的字符数目超过256个，char型无法表示。**

**wchar\_t数据类型一般为16位或32位，但不同的C或C++库有不同的规定，如GNU Libc规定wchar\_t为32位，总之，wchar\_t所能表示的字符数远超char型。**

**int err = fopen\_s(&fp, "D:\\a.txt", "r");**

**fread()**

**函数原型：  
size\_t fread ( void \*buffer, size\_t size, size\_t count, FILE \*stream) ; 返回真实读取的项数**

**参数1：buffer  
用于接收数据的内存地址**

**参数2：size  
要读的每个数据项的字节数，单位是字节**

**参数3：count  
要读count个数据项，每个数据项size个字节.**

**参数4：stream  
输入流**

**char buffer;**

**int size= fread(&buffer, sizeof(char), 1, fp);**

**while (size)**

**{**

**cout << buffer;**

**size = fread(&buffer, sizeof(char), 1, fp);**

**}**

**fclose()**

**函数原型： int fclose( FILE \*fp );  
如果流成功关闭，fclose 返回 0，否则返回EOF（-1）；**

**fclose(fp);**

**fwrite（）  
函数原型  
size\_t fwrite(const void\* buffer, size\_t size, size\_t count, FILE\* stream);  
参数与fread一样这里buffer是写入的内容**

**err = fopen\_s(&fp, "D:\\a.txt", "a+");**

**buffer = 'e';**

**fwrite(&buffer,sizeof(char),1,fp);**

**fclose(fp);**

**附加一个**

**fseek（）**

**函数原型  
int fseek(FILE \*stream, long offset, int fromwhere); 函数设置文件指针stream的位置。**

**成功，返回0，失败返回-1，并设置error的值，可以用perror()函数输出错误。**

**参数1：stream为文件指针**

**参数2：offset为偏移量，正数表示正向偏移，负数表示负向偏移**

**参数3：fromwhere设定从文件的哪里开始偏移  
参数3选项：  
0 ： SEEK\_SET： 文件开头  
1 ： SEEK\_CUR： 当前位置  
2 ： SEEK\_END： 文件结尾**

**fseek(fp,100L,0); 把stream指针移动到离文件开头100字节处；**

**fseek(fp,100L,1); 把stream指针移动到离文件当前位置100字节处；**

**fseek(fp,-100L,2); 把stream指针退回到离文件结尾100字节处。**

**偏移量后加 L，表示 long int 型。  
其实，不加L，一般编译器也都知道是 long 型**

**于是自己写的代码：**

**4fio.c**

**#include<stdio.h>**

**#include<string.h>**

**#include <fcntl.h>**

**#include<stdlib.h>**

**int main(int argc,char\* argv[])**

**{**

**FILE \*fp;**

**char msg;**

**char msg2[100];**

**int err;**

**if(argc<=1)**

**{**

**printf("Usage:%s filenane\a\n",argv[0]);**

**exit(1);**

**}**

**fp = fopen(argv[1],"at+");**

**if(!fp)**

**{**

**printf("fail to open!Check filename\n");**

**exit(1);**

**}**

**printf("%s content: ",argv[1]);**

**err = fread(&msg,1,1,fp);**

**while(err)**

**{**

**printf("%c",msg);**

**err = fread(&msg,1,1,fp);**

**}**

**while(1)**

**{**

**printf("\nWrite something to file %s\nor end: 0",argv[1]);**

**scanf("%s",msg2);**

**err = fwrite(msg2,strlen(msg2),1,fp);**

**if(!err)**

**{**

**printf("write err!");**

**exit(1);**

**}**

**printf("now %s content: ",argv[1]);**

**fseek(fp,0,0);**

**err = fread(&msg,1,1,fp);**

**while(err)**

**{**

**printf("%c",msg);**

**err = fread(&msg,1,1,fp);**

**}**

**if(msg2[0]=='0')**

**{**

**break;**

**}**

**}**

**fclose(fp);**

**return 0;**

**}**



**实验总结：**

**对文件的操作基于数据流的文件 I/O 操作其实算比较熟悉的，在这里看了示例之后有了更深的理解，然后写个小程序演示了了一下，需要注意的是一些参数使用时的形态比如指针，还有一些打开文件可能出现异常需要处理一下。一些：`fopen`是C标准函数，因此拥有良好的移植性；而`open`是UNIX系统调用，移植性有限，如果从文件IO的角度来看，前者属于低级IO函数，后者属于高级IO函数。低级和高级的简单区分标准是：谁离系统内核更近。低级文件IO运行在内核态，高级文件IO运行在用户态。使用fopen函数，由于在用户态下就有了缓冲，因此进行文件读写操作的时候就减少了用户态和内核态的切换（切换到内核态调用还是需要调用系统调用API:read，write）；而使用open函数，在文件读写时则每次都需要进行内核态和用户态的切换；表现为，如果顺序访问文件，fopen系列的函数要比直接调用open系列的函数快；如果随机访问文件则相反。**

**学号：031602504姓名：陈俊杰 学院：数学与计算机科学学院**

**专业：网络工程 班级：计算机5班**

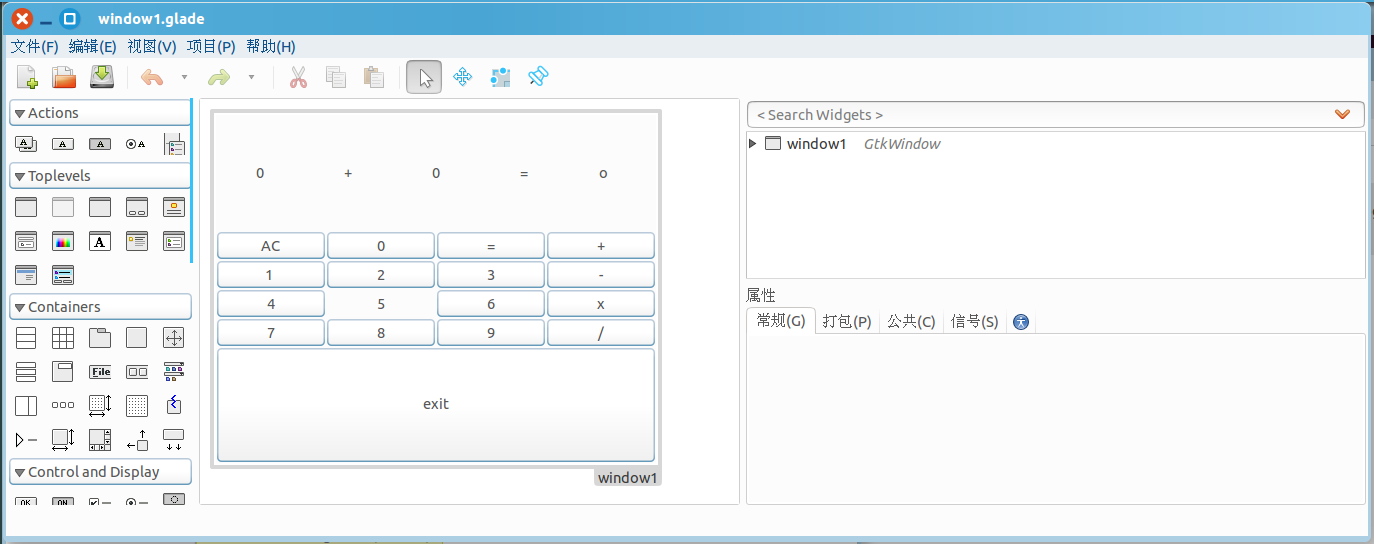
**《Linux 操作系统设计实践》**

**实验五：图形界面**

**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

**在实验室拷的ubuntukylin-14.04，其自带的了gtk3.0，glade2.0，使用学习到的make用法进行编译。利用glade创建一个计算机的ui界面，如图**

**保存成window1.glade在同一目录创建cal.c，Makefile；**

**Makefile内容：**

**cal.exe: cal.c**

**gcc cal.c -o cal.exe `pkg-config --cflags --libs gtk+-3.0 libglade-2.0`**

**clean:**

**rm -f \*.o outFile**

**cal.c内容：**

**#include <glade/glade.h>**

**#include <gtk/gtk.h>**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**void on\_delete\_event (GtkWidget \*widget,GdkEvent\* event,gpointer data);**

**void on\_number\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_AC\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_equ\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_add\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_sub\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_mul\_clicked(GtkWidget \*button,gpointer userdata);**

**void on\_div\_clicked(GtkWidget \*button,gpointer userdata);**

**GtkWidget \*label[6];**

**GtkWidget \*numberbutton[11];**

**int sign = 0;int num=0;**

**int main(int argc ,char \* argv[])**

**{**

**GtkBuilder \*builder;**

**GtkWidget \*window;**

**GtkWidget \*exitbutton;**

**GtkWidget \*ACbutton;**

**GtkWidget \*equbutton;**

**GtkWidget \*addbutton;**

**GtkWidget \*subbutton;**

**GtkWidget \*mulbutton;**

**GtkWidget \*divbutton;**

**int i;**

**gtk\_init (&argc, &argv);**

**builder = gtk\_builder\_new ();**

**gtk\_builder\_add\_from\_file (builder, "window1.glade", NULL);**

**window = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "window1"));**

**exitbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "exit"));**

**ACbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "AC"));**

**equbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "equ"));**

**addbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "add"));**

**subbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "sub"));**

**mulbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "mul"));**

**divbutton = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "div"));**

**for(i=0;i<=9;i++)**

**{**

**char number[10] = "number";**

**char id[10]="0";**

**id[0]=i+48;**

**strcat(number,id);**

**numberbutton[i] = GTK\_WIDGET (gtk\_builder\_get\_object (builder,number));**

**num = i;**

**g\_signal\_connect(G\_OBJECT(numberbutton[i]),"clicked",G\_CALLBACK(on\_number\_clicked),(gpointer)num);**

**}**

**for(i=0;i<=5;i++)**

**{**

**char \_label[10] = "label";**

**char id[10]="0";**

**id[0]=i+48;**

**strcat(\_label,id);**

**label[i] = GTK\_WIDGET (gtk\_builder\_get\_object (builder,\_label));**

**}**

**gtk\_builder\_connect\_signals (builder, NULL);**

**g\_signal\_connect(G\_OBJECT(window),"delete\_event",G\_CALLBACK(on\_delete\_event),NULL);**

**g\_signal\_connect(G\_OBJECT(exitbutton),"clicked",G\_CALLBACK(on\_delete\_event),NULL);**

**g\_signal\_connect(G\_OBJECT(ACbutton),"clicked",G\_CALLBACK(on\_AC\_clicked),NULL);**

**g\_signal\_connect(G\_OBJECT(equbutton),"clicked",G\_CALLBACK(on\_equ\_clicked),NULL);**

**g\_signal\_connect(G\_OBJECT(addbutton),"clicked",G\_CALLBACK(on\_add\_clicked),NULL);**

**g\_signal\_connect(G\_OBJECT(subbutton),"clicked",G\_CALLBACK(on\_sub\_clicked),NULL);**

**g\_signal\_connect(G\_OBJECT(mulbutton),"clicked",G\_CALLBACK(on\_mul\_clicked),NULL);**

**g\_signal\_connect(G\_OBJECT(divbutton),"clicked",G\_CALLBACK(on\_div\_clicked),NULL);**

**g\_object\_unref (G\_OBJECT (builder));**

**gtk\_widget\_show\_all(window);**

**gtk\_main ();**

**return 0;**

**}**

**void on\_delete\_event (GtkWidget \*widget,GdkEvent\* event,gpointer data)**

**{**

**gtk\_main\_quit();**

**}**

**void on\_number\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**int n = (int)userdata;**

**const char \*first = gtk\_label\_get\_text(GTK\_LABEL(label[0]));**

**const char\* second = gtk\_label\_get\_text(GTK\_LABEL(label[2]));**

**char this[20]={0};**

**if(sign==0)**

**{sprintf(this,"%s",first);}**

**else if(sign==1)**

**{sprintf(this,"%s",second);}**

**this[strlen(this)]='\0';**

**char nthis[5];**

**sprintf(nthis,"%d",n);**

**strcat(this,nthis);**

**if(sign==0)**

**{gtk\_label\_set\_text(GTK\_LABEL(label[0]),this);}**

**else if(sign ==1)**

**{gtk\_label\_set\_text(GTK\_LABEL(label[2]),this);}**

**}**

**void on\_AC\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char buf[10]="0";**

**int i;**

**for(i=0;i<=4;i++)**

**{**

**if(i!=3)**

**{**

**gtk\_label\_set\_text(GTK\_LABEL(label[i]),buf);**

**}**

**}**

**sign = 0;**

**}**

**void on\_add\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char op[10]="+";**

**gtk\_label\_set\_text(GTK\_LABEL(label[1]),op);**

**sign = 1;**

**}**

**void on\_sub\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char op[10]="-";**

**gtk\_label\_set\_text(GTK\_LABEL(label[1]),op);**

**sign = 1;**

**}**

**void on\_mul\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char op[10]="x";**

**gtk\_label\_set\_text(GTK\_LABEL(label[1]),op);**

**sign = 1;**

**}**

**void on\_div\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char op[10]="/";**

**gtk\_label\_set\_text(GTK\_LABEL(label[1]),op);**

**sign = 1;**

**}**

**void on\_equ\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**const char\* first = gtk\_label\_get\_text(GTK\_LABEL(label[0]));**

**const char\* op = gtk\_label\_get\_text(GTK\_LABEL(label[1]));**

**const char\* second = gtk\_label\_get\_text(GTK\_LABEL(label[2]));**

**int nfirst = atoi(first);**

**int nsecond = atoi(second);**

**int result=0;**

**if(op[0]=='+')**

**{**

**result = nfirst+nsecond;**

**}**

**else if(op[0]=='-')**

**{**

**result = nfirst-nsecond;**

**}**

**else if(op[0]=='x')**

**{**

**result = nfirst\*nsecond;**

**}**

**else if(op[0]=='/')**

**{**

**if(nsecond==0)**

**{**

**char tip[50] = "second number should not be zero!";**

**printf("%s\n",tip);**

**gtk\_label\_set\_text(GTK\_LABEL(label[4]),tip);**

**}**

**result = nfirst/nsecond;**

**}**

**char strresult[10];**

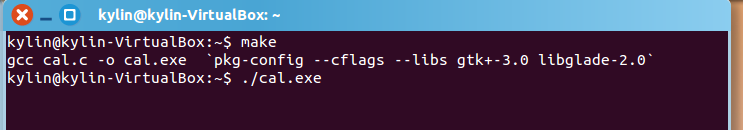
**sprintf(strresult,"%d",result);**

**gtk\_label\_set\_text(GTK\_LABEL(label[4]),strresult);**

**sign = 0;**

**}**

**运行结果：**





**实验总结：**

**注意点，查了好多的资料地方才终于找到的，用来解决碰到的问题：**

**Makefile：**

Makefile 是和 make 命令一起配合使用的.

很多大型项目的编译都是通过 Makefile 来组织的, 如果没有 Makefile, 那很多项目中各种库和代码之间的依赖关系不知会多复杂.

格式：target ... : prerequisites ... **\*注\*** command太长, 可以用 "\" 作为换行符

command

最简单的makefile文件

calc: main.c getch.c getop.c stack.c

gcc -o calc main.c getch.c getop.c stack.c

第一行冒号之前的calc，我们称之为目标（target），被认为是这条语句所要处理的对象，具体到这里就是我们所要编译的这个程序calc。冒号后面的部分（main.c getch.c getop.c stack.c），我们称之为依赖关系表，也就是编译calc所需要的文件，这些文件只要有一个发生了变化，就会触发该语句的第三部分，我们称其为命令部分，相信你也看得出这就是一条编译命令。现在我们只要将上面这两行语句写入一个名为Makefile或者makefile的文件，然后在终端中输入make命令，就会看到它按照我们的设定去编译程序了

**命令行下编译出现Gtk-CRITICAL \*\*: IA\_\_gtk\_widget\_show\_all: assertion `GTK\_IS\_WIDGET (widget)' failed：**

这个问题一般很少见，路径问题引起的，最后研究后发现，必须写绝对路径才行，比如：gtk\_builder\_add\_from\_file(builder,"/home/lmm/文档/login.glade", NULL )中glade文件位置改为绝对路径

**gtk\_widget\_show\_all: assertion `GTK\_IS\_WIDGET (widget)' failed：**

这是一个比较常见的问题，一般是因为这句代码引起的：window = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "window"))，原因是在利用glade3构造GUI时，用户所起的主窗口名字与默认的不一样。代码中的"window"是默认的，而用户可能起得名字是"mainWindow"。

解决方案：将构建名称改为自己所起的名字，例如：window = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "mainWindow"))

**GLib-GObject-ERROR \*\*: cannot create instance of abstract (non-instantiatable) type `GtkBox'：**

这是因为GTK+版本与glade3不兼容产生的，galde3所需要的库是GTK+3.0，而不是GTK+2.0，网上出现的编译命令一般为是gcc -o main main.c `pkg-config --cflags --libs gtk+-2.0 gmodule-export-2.0`。

解决方案：将命令改为gcc -o main main.c `pkg-config --cflags --libs gtk+-3.0 gmodule-export-2.0`

**一些了解到的内容：**

**g\_signal\_connect 宏**有 4 个参数：连接的对象，就是要连接信号的控件的指针(注意：必须是已创建完的控件的指针)， 需要用 G\_OBJECT 宏来转换，如G\_OBJECT(window)；；信号名称，就是要连接的信号名称，为字符串形式，用双引号引起来。不同的控件拥 有的信号名称是不一样的，如window窗口控件的信号："delete\_event"；回调函数，指信号发生时调用的函数，这里只用到函数名称，需要用 G\_CALLBACK 宏来转换一下，如G\_CALLBACK(gtk\_main\_quit)；传递给回调函数的参数，它的值类型应该为 gpointer。如果不是这一类型可以强制转换， 如果没有参数则为 NULL。这里只能传递一个参数，如果有多个参数，可以先将多个参数 定义为一个结构，再将此结构作为参数传递过来。

这次实验学会了实验glade进行界面设计，通过gtk链接使用glade创建的界面，然后是连接信号的控件，基本的控件创建与使用，布局等因为是用glade创建的界面所以是没怎么涉及到，总之这次实验学习基于 GTK+的界面编程开发方法是很有收获的，也可以在实验二、三或四的基础上实现增加图形界面功能后的展 示或操作不过要写挺多的。

**学号：031602504姓名：陈俊杰 学院：数学与计算机科学学院**

**专业：网络工程 班级：计算机5班**

**《Linux 操作系统设计实践》**

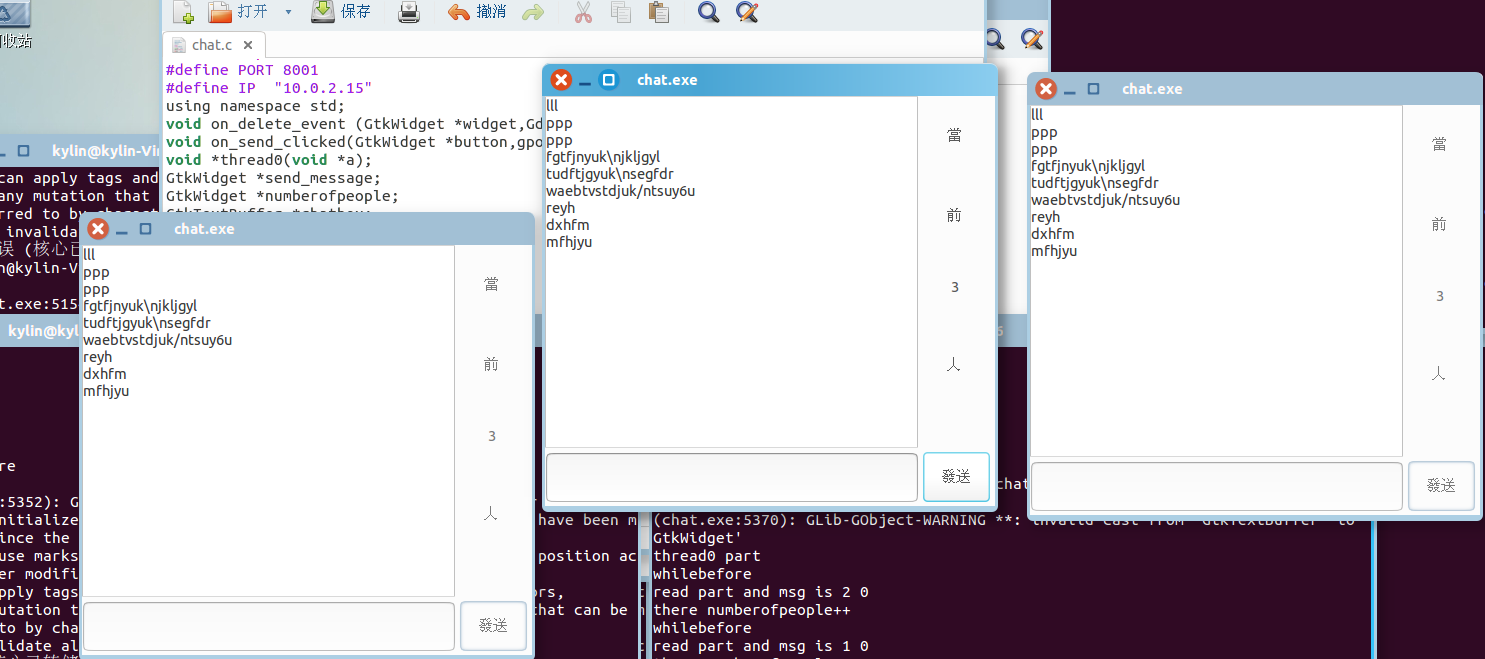
**实验六：综合应用**

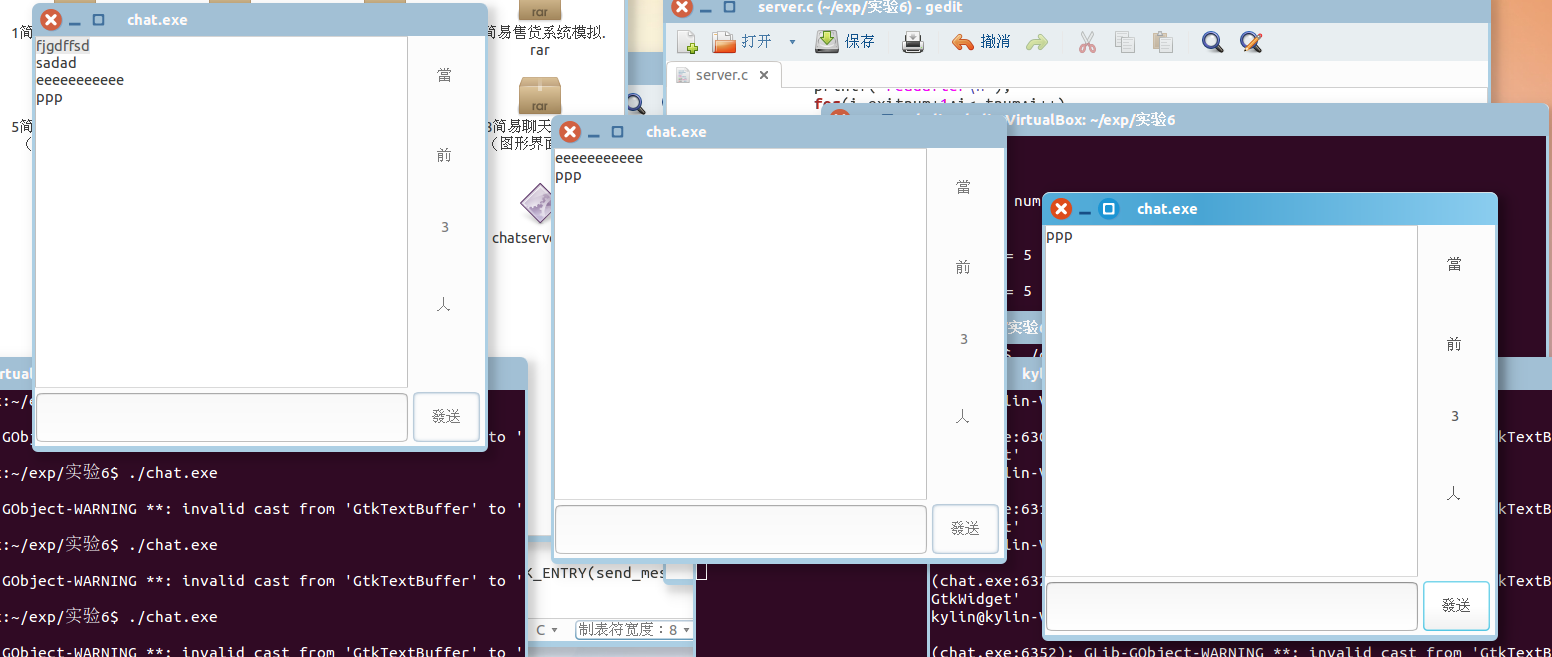
**实验环境：**

**ubuntukylin-14.04**

**实验内容：**

**结果图，简单的多人匿名聊天**

****

****

**代码：**

**chat.c**

**#include <glade/glade.h>**

**#include <gtk/gtk.h>**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**#include<iostream>**

**#include<sys/socket.h>**

**#include<netinet/in.h>**

**#include <arpa/inet.h>**

**#include <pthread.h>**

**#define PORT 8001**

**#define IP "10.0.2.15"**

**using namespace std;**

**void on\_delete\_event (GtkWidget \*widget,GdkEvent\* event,gpointer data);**

**void on\_send\_clicked(GtkWidget \*button,gpointer userdata);**

**void \*thread0(void \*a);**

**GtkWidget \*send\_message;**

**GtkWidget \*numberofpeople;**

**GtkTextBuffer \*chatbox;**

**int sockfd,newfd;**

**struct sockaddr\_in server;**

**int num =0;**

**int main(int argc,char\* argv[])**

**{**

**sockfd = socket(AF\_INET,SOCK\_STREAM,0);**

**memset(&server,'0',sizeof(struct sockaddr\_in));**

**server.sin\_family=AF\_INET;**

**server.sin\_port=htons(PORT);**

**server.sin\_addr.s\_addr = inet\_addr(IP);**

**if(connect(sockfd, (struct sockaddr \*) &server,sizeof(server))==-1)**

**{printf("connect error!\n");}**

**//定義組建**

**GtkBuilder \*builder;**

**GtkWidget \*window;**

**GtkWidget \*send;**

**gtk\_init (&argc, &argv);**

**//組建從glade獲取**

**builder = gtk\_builder\_new ();**

**gtk\_builder\_add\_from\_file (builder, "chat.glade", NULL);**

**window = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "window1"));**

**send = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "button1"));**

**chatbox = (GtkTextBuffer\*)GTK\_WIDGET (gtk\_builder\_get\_object (builder, "Chatbox"));**

**send\_message = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "entry1"));**

**numberofpeople = GTK\_WIDGET (gtk\_builder\_get\_object (builder, "number"));**

**//信號鏈接**

**g\_signal\_connect(G\_OBJECT(window),"delete\_event",G\_CALLBACK(on\_delete\_event),NULL);**

**g\_signal\_connect(G\_OBJECT(send),"clicked",G\_CALLBACK(on\_send\_clicked),NULL);**

**//創建線程**

**pthread\_t id;**

**pthread\_create(&id,NULL,thread0,NULL);**

**//顯示**

**gtk\_widget\_show\_all(window);**

**gtk\_main ();**

**return 0;**

**}**

**void on\_delete\_event (GtkWidget \*widget,GdkEvent\* event,gpointer data)**

**{**

**char temp[3]="ab";**

**temp[0]=1;**

**write(sockfd,temp,strlen(temp));**

**sleep(1);**

**gtk\_main\_quit();**

**}**

**void on\_send\_clicked(GtkWidget \*button,gpointer userdata)**

**{**

**char msg[1024]="";**

**const char \*msg\_ = gtk\_entry\_get\_text(GTK\_ENTRY(send\_message));**

**sprintf(msg,"%s",msg\_);**

**if(strlen(msg)==0)**

**{return;}**

**if(strlen(msg)>1024)**

**{**

**return;**

**}**

**if(write(sockfd,msg,strlen(msg))==-1)**

**{**

**printf("sendmsg error \n");**

**}**

**gtk\_entry\_set\_text(GTK\_ENTRY(send\_message),"");**

**}**

**void \*thread0(void \*a)**

**{**

**while(1)**

**{**

**char newmsg[1024];**

**read(sockfd,newmsg,1024);**

**if(strlen(newmsg)>2&&newmsg[strlen(newmsg)-2]==29)**

**{**

**newmsg[strlen(newmsg)-2]='\0';**

**gtk\_label\_set\_text(GTK\_LABEL(numberofpeople),newmsg);**

**continue;**

**}**

**strcat(newmsg,"\n");**

**GtkTextIter start, end;**

**gtk\_text\_buffer\_get\_end\_iter(chatbox,&end);**

**gtk\_text\_buffer\_get\_start\_iter(chatbox,&start);**

**gtk\_text\_buffer\_insert(chatbox,&end,newmsg,strlen(newmsg));**

**}**

**}**

**server.c**

**#include <glade/glade.h>**

**#include <gtk/gtk.h>**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**#include<iostream>**

**#include<sys/socket.h>**

**#include<netinet/in.h>**

**#include <netdb.h>**

**#include <sys/types.h>**

**#include <arpa/inet.h>**

**#include <pthread.h>**

**#define PORT 8001**

**int sockfd,newfd[200];**

**int new\_[200];**

**int tnum,exitnum=0;**

**struct sockaddr\_in client[100];**

**void \* thread0(void \* arg)**

**{**

**int i;**

**int \_num = \*(int \*)arg;**

**printf("thread0 part \n");**

**while(1)**

**{**

**char msg[1024]="";**

**printf("readbefore and \_num = %d\n",\_num);**

**read(newfd[\_num],msg,1024);**

**if(msg[0]==1)**

**{**

**exitnum++;**

**printf("exitnum=%d\n",exitnum);**

**char temp1[3]="10";**

**temp1[0]=29;**

**char temp0[18] = "";**

**sprintf(temp0,"%d",tnum-exitnum);**

**strcat(temp0,temp1);**

**new\_[\_num]=0;**

**for(i=1+exitnum;i<=tnum;i++)**

**{ if(new\_[i]==1)**

**{**

**write(newfd[i],temp0,30);**

**printf("writeto %d\n",tnum);**

**}**

**}**

**break;**

**}**

**printf("readafter\n");**

**for(i=exitnum+1;i<=tnum;i++)**

**{ if(new\_[i]==1)**

**{**

**write(newfd[i],msg,1024);**

**}**

**}**

**}**

**}**

**int main()**

**{**

**int num=0;**

**pthread\_t id[100];**

**int i;**

**for(i=0;i<200;i++)**

**{new\_[i]=1;}**

**struct sockaddr\_in server;**

**sockfd = socket(AF\_INET,SOCK\_STREAM,0);**

**memset(&server,'0',sizeof(struct sockaddr\_in));**

**memset(client,'0',sizeof(struct sockaddr\_in)\*100);**

**server.sin\_family=AF\_INET;**

**server.sin\_port=htons(PORT);**

**server.sin\_addr.s\_addr = htonl(INADDR\_ANY);**

**socklen\_t slen = sizeof(struct sockaddr\_in);**

**if(bind(sockfd , (struct sockaddr \*) &server, sizeof(struct sockaddr))==-1)**

**{printf("bind error\nwait moments\n");}**

**if(listen(sockfd,100))**

**{printf("listen error\n");}**

**while(1)**

**{**

**printf("accept before\n");**

**num++;**

**newfd[num] = accept(sockfd,(struct sockaddr \*) &client[num],&slen);**

**printf("accept part\n");**

**char temp1[3]="10";**

**temp1[0]=29;**

**char temp0[18] = "";**

**sprintf(temp0,"%d",num-exitnum);**

**strcat(temp0,temp1);**

**printf("now temp0 is %s\n",temp0);**

**for(i=exitnum+1;i<=num;i++)**

**{ if(new\_[i]==1)**

**{**

**write(newfd[i],temp0,strlen(temp0));**

**printf("send end\n");**

**}**

**}**

**tnum=num;//防止主線程先行把num給+1;**

**printf("before enter pthread num is %d\n",tnum);**

**pthread\_create(&id[num],NULL,thread0,&tnum);**

**}**

**close(sockfd);**

**return 0;**

**}**

**编译：**

**Makefile：**

**all:chat.exe chatserver.exe**

**chat.exe: chat.c**

**g++ chat.c -o chat.exe `pkg-config --cflags --libs gtk+-3.0 libglade-2.0` -lpthread**

**chatserver.exe:server.c**

**g++ server.c -o chatserver.exe `pkg-config --cflags --libs gtk+-3.0 libglade-2.0` -lpthread**

**clean:**

**rm -f \*.o outFile**

**编译：make all**

**网络编程结合界面设计给一个简单的聊天程序包装**

**一个服务器端，一个客户端，客户端发消息到服务器，服务器返回到每个客户端的聊天窗口，实现多人的相互交流。这里用到线程控制，互斥量的理解，进程（线程）调度的理解，还有网络编程的基础及界面设计的知识。那么直接上代码。有一个过程中因为客户端关闭时所有界面都会消失，因为没有进行相应处理，群发会有异常，在看了pdf里的聊天室后，解决办法是在服务器关是发信号过去，在服务器端把描述字套接字外加连接状态打包成结构，在群发的时候根据连接转态选择发不发，人数变化同样在关闭前发送修改信号到服务器端，总的来说由于一开始考虑不周到而造成的异常，在知道解决办法后，由于修改的地方比较明确 ，在不在整体的修改上加了个对应数组来当连接转态，客户端在关闭时发要关的信号使服务器接受到信息做相应修改转态不去群发，于是算是解决了；**

**实验总结：**

**这次的综合实验总体花的时间有点多，因为是断断续续写的，每次接着写都要重新看下之前写的，虽然大致思路是一样的，但时间是多一些，然后那些网络编程在实验3打完之后实际是忘得差不多了，所以又重新学了一遍网络编程的内容，同时断断续续每次相当于都复习理解那些东西的概念，所以现在来说对套接字和那些函数的应用计tcp的连接过程有了较有印象的记忆，收获很大，然后界面设计实际上花的实际不是很多，因为glade设计好界面，在代码中直接连接用，少了定义及布局的程序简单得多，而且在glade里面还发现可以直接在信号的地方写绑定的函数，如果直接写那的话，实际客户端的程序就只需要考虑功能的实现了。然后由于开始的思路不完整没思考到线程的数据相关，以及对数据的安排不是很适当导致其容易崩坏，bug最终修修补补总算是填上了，总的来说对Linux的理解加深，对网络编程可以说有了很大的进步，界面设计也学会了应用glade简单设计。**