桂林电子科技大学

**实验八 综合实例**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **实验8 综合实例** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计算机科学与技术 | | |
| 学 号 | 1600300128 | | | 姓名 | | 王涛 | | |
| 实验日期 | 2018 | 年 | 1 | | 月 | | 5 | 日 |
|  |  | | | | | | | |

## 实验目的

1、 掌握Linux系统下进行综合应用开发的基本方法；

2、 掌握Linux系统下进行综合应用开发的合适工具选型和应用；

3、 掌握在Linux应用开发中获取资料和在线帮助的方法。

## 实验要求

通过一个应用实例说明开发的基本方法、工具的选择和应用。

## 实验步骤

一、编辑、调试、运行以下程序

1 pipe.sh

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | # 创建管道文件  if [ -e "myfifo" ]  then  rm ./myfifo  mkfifo ./myfifo  else  mkfifo ./myfifo  fi  clear  # 在死循环中不停的读取管道内容，读完之后根据读到的内容进行操作，然后写回管道中  #resp=""  while true  do  r=$(tr -d '\0' < myfifo)  case "$r" in  1)  resp=`ls` ;;  2)  resp=`date` ;;  3)  resp=`uname -a` ;;  \*)  resp="你的输入有误" ;;  esac  echo $resp > myfifo  echo -e "Sent [ "$resp" ] to fifo\n"  done |

2 server.c 服务器端程序

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152 | #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <unistd.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <sys/socket.h>  #include <pthread.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <semaphore.h>  #define P sem\_wait  #define V sem\_post  #define mutex &mutex\_real  #define BUFSIZE 200  int sockfd;//服务器socket  int fds[100];//客户端的socketfd,100个元素，fds[0]~fds[99]  int size =100 ;//用来控制进入客户端个数  char\* IP = "0.0.0.0";  short PORT = 10222;  typedef struct sockaddr SA;  int pipe\_fd;  sem\_t mutex\_real;  //初始化socket  void init(){  //创建socket获得描述符  sockfd = socket(PF\_INET,SOCK\_STREAM,0);  if (sockfd == -1){  perror("创建socket失败");  exit(-1);  }  //下面两行设置 socket 可以重用地址，这样在使用 Ctrl + C 终止 server 后不用等待即可再次创建 server  int mw\_optval = 1;  setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR,  (char \*)&mw\_optval,sizeof(mw\_optval));    //设置连接的参数，包括地址，端口，连接类型  struct sockaddr\_in addr;  addr.sin\_family = PF\_INET;  addr.sin\_port = htons(PORT);  addr.sin\_addr.s\_addr = inet\_addr(IP);  //绑定地址  if (bind(sockfd,(SA\*)&addr,sizeof(addr)) == -1){  perror("绑定失败");  exit(-1);  }  //监听端口  if (listen(sockfd,100) == -1){  perror("设置监听失败");  exit(-1);  }  }  //每一次连接会创建一个这样子的线程进行处理  void\* service\_thread(void\* p){  //获得连接套接字的描述符  int fd = \*(int\*)p;  int readbytes;  while(1){  char rqst[BUFSIZ]; //从socket接收到的字符串  char resp[BUFSIZ]; //从管道文件读出来的字符串    //接收消息，如果连接断开就说明退出了，释放描述符在数组中的位置  if (recv(fd,rqst,BUFSIZ,0) <= 0){  int i;  for (i = 0;i < size;i++){  if (fd == fds[i]){  fds[i] = 0;  break;  }  }  printf("fd = %d\t连接断开\n",fd);  pthread\_exit(0);  }  //对管道的操作是互斥的，管道文件只有一对，所以每次只能有一个线程跟它们交互  P(mutex);  pipe\_fd = open("myfifo", O\_WRONLY); //只写模式打开管道文件  write(pipe\_fd,rqst, BUFSIZ); //把接收到的字符串写到管道文件  close(pipe\_fd); //关闭管道文件  pipe\_fd = open("myfifo", O\_RDONLY); //只读模式打开管道文件  readbytes = read(pipe\_fd, resp, BUFSIZ); //从管道文件中读取内容  close(pipe\_fd);  V(mutex);  //把服务器接受到的信息发回给该客户端  resp[readbytes] = '\0';  send(fd,resp,strlen(resp),0);  }  }  void service(){  printf("服务器启动\n");  //初始化互斥信号量  sem\_init(mutex, 0, 1);  while(1){  struct sockaddr\_in fromaddr;  socklen\_t len = sizeof(fromaddr);  //接受连接  int fd = accept(sockfd,(SA\*)&fromaddr,&len);  if (fd == -1){  printf("客户端连接出错...\n");  continue;  }  //从数组中找到一个没被占用的位置，把连接套接字的描述符存进去  int i = 0;  for (i = 0;i < size;i++){  if (fds[i] == 0){  //记录客户端的socket  fds[i] = fd;  printf("接受一个连接：fd = %d\n",fd);  //有客户端连接之后，启动线程给此客户服务  pthread\_t tid;  pthread\_create(&tid,0,service\_thread,&fd);  break;  }  }  //如果数组100个位置都被占用，说明此时已经存在100个用户  if (size == i){  char\* str = "已经达到100个用户连接上限!";  send(fd,str,strlen(str),0);  close(fd);  }  }  }  int main(void){  init();  service();  return 0;  } |

gcc -o server server.c -lpthread

3 client.c 客户端程序

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <pthread.h>  #define BUFSIZE 200  int sockfd;//客户端socket  char \* IP = "192.168.150.129";//服务器的IP  short PORT = 10222;//服务器服务端口  typedef struct sockaddr SA;  //初始化  void init() {  //创建  sockfd = socket(PF\_INET, SOCK\_STREAM, 0);  struct sockaddr\_in addr;  //设置参数  addr.sin\_family = PF\_INET;  addr.sin\_port = htons(PORT);  addr.sin\_addr.s\_addr = inet\_addr(IP);  //进行连接  if (connect(sockfd, (SA\*)&addr, sizeof(addr)) == -1) {  perror("无法连接到服务器");  exit(-1);  }  printf("客户端启动成功\n\*输入1可以查看服务端当前目录信息\n\*输入2可以查看当前服务端上的时间\n\*输入3可以查看服务端的系统版本\n");  }  void start() {  pthread\_t id;    //创建一个线程进行接收  void\* recv\_thread(void\*);  pthread\_create(&id, 0, recv\_thread, 0);  char buf[BUFSIZ];  //将用户的输入发给服务端  while (1) {    scanf("%s", buf);  send(sockfd, buf, strlen(buf), 0);  }  close(sockfd);  }  void\* recv\_thread(void\* p) {  int numbytes;  char buf[BUFSIZ];  while((numbytes = recv(sockfd, buf, BUFSIZ, 0)) >0)  {  buf[numbytes] = '\0';  printf("%s\n", buf);  }  }  int main(void) {  init();  start();  return 0;  } |

gcc -o client client.c -lpthread

4、 开启三个终端，分别运行管道shell、服务器端程序、客户端程序。

首先运行pipe.sh，然后编译运行server.c，最后编译运行client.c

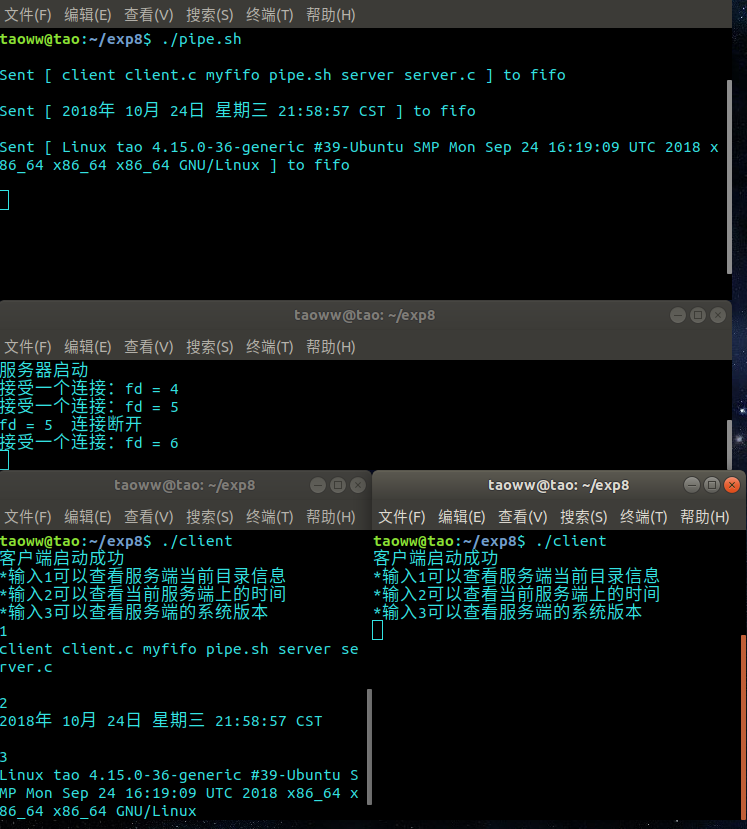
在客户端根据提示分别输入相应的数字，观察实验结果，并对实验结果进行解析。



分析：首先开启pipe.sh，它的作用是循环读取管道中的内容，读取到内容只够根据内容，执行不同的操作将结果写入管道。之后开启server.c，他的作用是创建套接字监听客户端连接，每连接一个客户端，就会创建一个线程来接收用户消息，不能超过最大连接数量，接收用户发送的请求之后将其写入管道，使用信号量达到对管道写入的互斥。之后读取管道内容发送给客户端。client.c的作用是创建客户端连接服务端，向服务端发消息。

所以，总的流程就是客户端连接服务端，向服务端发送操作命令，服务端将操作命令写入管道，pipe.sh读取操作命令执行相应操作，将操作结果写回管道，服务端读取管道内容将结果返回给客户端。

5、再开启第二个客户端，观察实验结果，与第4步有何不同，并对实验结果进行解析。



分析：在一个客户端没有收到回复之前，另一个客户端的请求需要等待，因为管道是互斥操作，一个客户端发送请求服务端写入管道，另一个客户端的请求需要等待。完成后再将下一个客户端的请求写入管道，完成相应的操作。

6、在本实例中用到了管道通信、socket通信，说明二者的适用性和局限性。

套接口可以说是网络编程中一个非常重要的概念，linux以文件的形式实现套接口，与套接口相应的文件属于sockfs特殊文件系统，创建一个套接口就是在sockfs中创建一个特殊文件，并建立起为实现套接口功能的相关数据结构。管道是单向的、先进先出的、无结构的字节流，它把一个进程的输出和另一个进程的输入连接在一起。写进程在管道的尾端写入数据，读进程在管道的首端读出数据。数据读出后将从管道中移走，其它读进程都不能再读到这些数据。管道提供了简单的流控制机制。进程试图读一个空管道时，在数据写入管道前，进程将一直阻塞。同样，管道已经满时，进程再试图写管道，在其它进程从管道中读走数据之前，写进程将一直阻塞。

7、在本实例中使用了线程同步机制semaphore信号量，通过man帮助学习sem\_wait和sem\_post两个函数原型，说明参数含义以及函数的功能。

函数sem\_post( sem\_t \*sem )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在阻塞，选择机制同样是由线程的调度策略决定的。函数sem\_wait( sem\_t \*sem )被用来阻塞当前线程直到信号量sem的值大于0，解除阻塞后将sem的值减一，表明公共资源经使用后减少。

8、根据你目前掌握的知识，说明设计实现一个图书信息管理系统，应考虑的主要方面有哪些？

（1）数据存储结构。图书信息管理系统核心的功能是书籍的增删改查、统计。少不了数据的操作，数据的存储简单的可以使用基本的数据结构，但是要实现数据的持久化保存需要引入数据库，设计相应的表结构，存储相应的数据。

（2）并发操作。一个系统肯定会出现并发操作的情况，这时候如何提高系统效率，保证系统的安全性正确性就是一个很重要的问题。从实验中使用最简单的多线程机制可以保证多个用户同时访问，但是对数据操作的需考虑到互斥性等，需要引入锁、信号量等机制。

（3）界面设计。好的系统肯定离不开优雅大气的UI设计，需要考虑到页面业务逻辑合理，用户操作体验，方便性等。同时对于软件的移动使用方便，可以考虑使用B/S架构，或者设计APP等。

（4）安全性。除了并发操作可能带来的数据错误，还要考虑到系统的安全性和健壮性，在系统中加入一些安全机制会很好的提高系统质量。

（5）完整的业务功能。系统的功能不要设计的反人类，功能要有用，无用的功能尽量减少。保证用尽量简单的方式完成具体的需求。

## 问题记录和实验总结

问题记录：

多线程c语言程序使用gcc编译报错

解决：使用了线程的c语言程序编译时需要加上-lpthread。

通过本次实验，学会了c语言的多线程的基本使用，也学会了简单的线程同步，通过样例代码能够理解线程间如何实现互斥的。通过网上的资料来了解我们的sem\_post(),sem\_wait()的使用，知道通过这两个函数可以实现堵塞的功能，来控制进程的同步。实验中通信方式使用了管道和socket，通过实验更加熟悉了这两种通信方式的使用和他们的应用场景及特点。