

源程序：puts，fputs，gets，fgets：P123

#include "apue.h"

int main(void)

{

/\*

\* puts与fputs：puts每次输出的时候带有换行符

\* gets与fgets：gets不安全，没有指定输入长度，会发生内存溢出问题；fgets每次 \*只读取n-1个字符，本道题n=2，每次只读取一个字符

\*/

char buf[2];

while(fgets(buf,2,stdin)!=NULL){

if(fputs(buf,stdout)==EOF)

printf("%s","error");

}

if(ferror(stdin))

printf("%s","error");

exit(0);

}

1. abcd
2. abcd
3. a

b

c

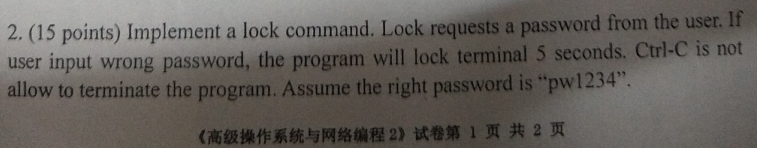
d

1. a

b

c

d



参考：P277

#include "apue.h"

//捕捉信号事件

static void sig\_int(int signo){

if(signo==SIGINT){

;

}

}

int main(void)

{

char c[1024];

while(scanf("%s",c)==1){

if(strcmp(c,"pw1234")==0){

printf("pass!\n");

}

else{

printf("Wrong password.\n");

sigset\_t newmask,oldmask;

sigemptyset(&newmask);

sigaddset(&newmask,SIGINT);//加入quit信号

if(signal(SIGINT,sig\_int)==SIG\_ERR){

printf("SIG ERROR\n");

}

if(sigprocmask(SIG\_BLOCK,&newmask,&oldmask)){//屏蔽quit信号

printf("SIG\_BLOCK ERROR\n");

exit(0);

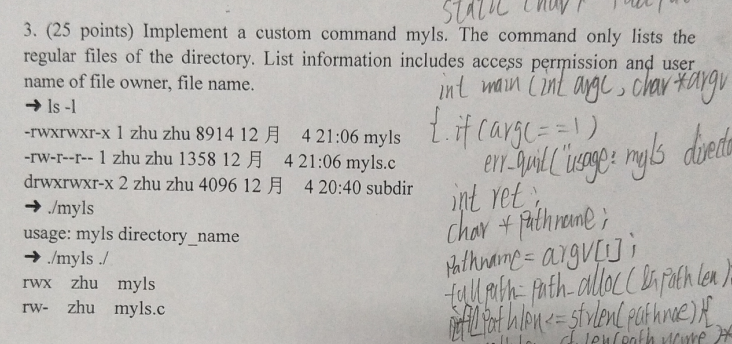
}

sleep(5);//锁住5秒

}

}

}



参考：

#include "apue.h"

#include <dirent.h>

#include <pwd.h>

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

{

if(argc==1){

printf("%s","usage:myls directory\_name");

exit(0);

}

DIR \*dir;

dir=opendir(argv[1]);//p104

struct dirent \* dirlist;

struct stat buf;

char permission[4]="---\0";

while(dirlist=readdir(dir)){

stat(dirlist->d\_name,&buf);//p74

mode\_t mode=buf.st\_mode;

if(S\_ISREG(mode)){

uid\_t uid;

uid=buf.st\_uid;

struct passwd \* pwd;

pwd=getpwuid(uid);//p144

char \* name= pwd->pw\_name;

if(mode & S\_IRUSR)

permission[0]='R';

if(mode & S\_IWUSR)

permission[1]='W';

if(mode & S\_IXUSR)

permission[2]='X';

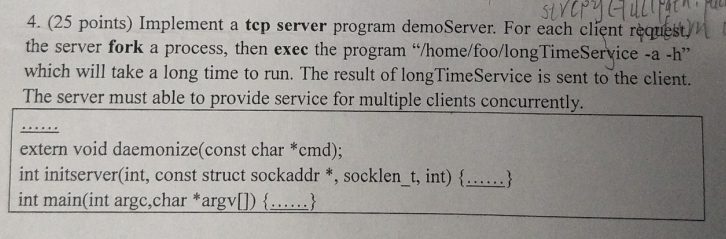
printf("%s%s%s%s%s\n",permission," ",name," ",

dirlist->d\_name);

}

}

}



参考：p497

#include <stdio.h>

#include <arpa/inet.h>

#include <syslog.h>

#include <signal.h>

#include <errno.h>

#include <fcntl.h>

#include <netdb.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/socket.h>

#define QLEN 10

#ifndef HOST\_NAME\_MAX

#define HOST\_NAME\_MAX 128

#endif

extern void daemonize(const char \* cmd);

int initserver(int ,struct sockaddr \* ,socklen\_t ,int );

void server(int sockfd);

void set\_cloexec(int);

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

{

struct addrinfo hint, \*ailist, \*aip;

int err, sockfd, n;

char \*host;

struct sockaddr\_in \*sinp;

const char \*addr;

char abuf[INET\_ADDRSTRLEN];

if(argc != 1)

{

fprintf(stderr,"usage:longTimeServer\n");

exit(1);

}

if((n=sysconf(\_SC\_HOST\_NAME\_MAX)) < 0)

n = HOST\_NAME\_MAX;

if((host=malloc(n)) == NULL)

{

perror("malloc");

exit(1);

}

if(gethostname(host,n) < 0)

{

perror("gethostname");

exit(1);

}

//hint内存空间分配

memset(&hint,0,sizeof(hint));

hint.ai\_flags = AI\_PASSIVE;

hint.ai\_socktype = SOCK\_STREAM;

hint.ai\_canonname = NULL;

hint.ai\_addr = NULL;

hint.ai\_next = NULL;

if((err=getaddrinfo(host,"ruptime",&hint,&ailist)) != 0)

{

fprintf(stderr,"getaddrinfo : %s\n",gai\_strerror(err));

exit(1);

}

for(aip=ailist; aip!=NULL; aip=aip->ai\_next)

{

if((sockfd=initserver(SOCK\_STREAM,aip->ai\_addr,aip->ai\_addrlen,QLEN)) >= 0)

{

server(sockfd);

exit(0);

}

}

exit(1);

}

/\*

\* 初始化服务:

\* (1)创建套接字socket()

\* (2)socket绑定地址bind()

\* (3)监听listen()

\* 返回：socket文件描述符

\*/

int initserver(int type, struct sockaddr \*addr,socklen\_t alen,int qlen)

{

int fd;

int err;

if((fd=socket(addr->sa\_family,type,0)) < 0)

return -1;

if(bind(fd,addr,alen) < 0)

{

perror("bind");

close(fd);

return -1;

}

if(type==SOCK\_STREAM || type==SOCK\_SEQPACKET)

{

if(listen(fd,qlen) < 0)

{

close(fd);

return -1;

}

}

return fd;

}

void server(int sockfd)

{

int clfd,status;

pid\_t pid;

set\_cloexec(sockfd);

while(1){

if((clfd=accept(sockfd,NULL,NULL))){

printf("%s","accept error");

exit(1);

}

if((pid==fork())<0){

printf("%s","fork error");

exit(1);

}else if(pid==0){

//fork()两次，防止僵尸进场

if((pid==fork())<0){

printf("%s","fork error");

exit(1);

}else if(pid>0){

//防止第一代孩子进程不退出

exit(1);

}

//执行longTimeServer

int result=execl("/home/foo/longTimeServer","longTimeServer","-a","-h",(char \*)0);

//发送运行结果到客户端

send(clfd,&result,sizeof(result),0);

close(clfd);

exit(1);

}else{//父亲进程

close(clfd);

waitpid(pid,&status,0);

}

}

}

//关闭文件描述符

void set\_cloexec(int sockfd)

{

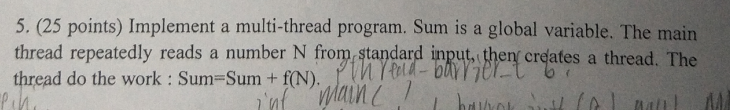
int val;

val = fcntl(sockfd,F\_GETFD);

val |= FD\_CLOEXEC;

fcntl(sockfd,F\_SETFD,val);

}



参考:p321互斥变量

#include <apue.h>

#include <stdio.h>

#include <pthread.h>

int total=0;

pthread\_mutex\_t mutex;

void \*tfn(void \* arg){

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

while(1){

//尝试锁住

if(pthread\_mutex\_trylock(&mutex)==0){

total+=num;

printf("%s","total:");

printf("%d\n",total);

//解锁

pthread\_mutex\_unlock(&mutex);

//退出循环

break;

}

}

}

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

{

pthread\_t tid;

void \* tret;

int num;

//初始化

pthread\_mutex\_init(&mutex,NULL);

while(scanf("%d",&num)==1){

pthread\_create(&tid,NULL,tfn,(void \*)&num);

}

//摧毁互斥变量

pthread\_mutex\_destroy(&mutex);

}