OPERATING SYSTEM

操作系统课程设计

张 柏 礼
bailey_zhang@sohu.com
东南大学计算机学院

1

设计内容

- 预备工作
- 实验一：Linux进程管理及其扩展
- 实验二： Shell的实现
- *实验三：文件系统的实现（选做）*

2

预备工作

- VirtualBox 安装
  - 或 VMware 安装
  - 或 不使用虚拟机直接安装Linux（双系统）
- Linux 的选择
  - Fedora
    - 服务器端
    - RedHat，CentOS 一个体系
  - Ubuntu
    - 桌面系统
    - Windows化（方便用户、过程黑盒）
- Linux的安装和配置
- C编译器 gcc

3

实验一：Linux进程管理及其扩展

- 实验目的：
  - 通过实验，加深理解进程控制块、进程队列等概念，了解进程管理的具体实施方法
- 实验内容：
  - 实现一个系统调用hide，使得可以根据指定的参数隐藏进程，使用户无法使用ps或top观察到进程状态。

4

实验一：Linux进程管理及其扩展

- **具体要求：**
  - **1)实现系统调用int hide(pid_t pid, int on)，在进程pid有效的前提下，如果on置1，进程被隐藏，用户无法通过ps或top观察到进程状态；如果on置0且此前为隐藏状态，则恢复正常状态。**
  - **2)考虑权限问题，只有root用户才能隐藏进程。**
  - **3)设计一个新的系统调用int hide_user_processes(uid_t uid, char *binname)，参数uid为用户ID号，当binname参数为NULL时，隐藏该用户的所有进程；否则，隐藏二进制映像名为binname的用户进程。该系统调用应与hide系统调用共存。**

5

实验一：Linux进程管理及其扩展

- **4)在/proc目录下创建一个文件/proc/hidden，该文件可读可写，对应一个全局变量hidden_flag，当hidden_flag为0时，所有进程都无法隐藏，即便此前进程被hide系统调用要求隐藏。只有当hidden_flag为1时，此前通过hide调用要求被屏蔽的进程才隐藏起来。**
- **5)在/proc目录下创建一个文件/proc/hidden_process，该文件的内容包含所有被隐藏进程的pid，各pid之间用空格分开。**

6

实验一：Linux进程管理及其扩展

- **实验指导：**
  - **隐藏进程一般有两种方法**
    - **(1) 正统方法：重载/替换 系统调用sys_getdents64()**
      - **/proc 是虚拟文件目录, 用来与操作系统内核进行交互，所有进程相关的静态、动态信息以/proc目录下文件形式发布**
      - **Linux系统用 sys_getdents64()来查询/proc目录下文件获取进程的相关信息**
      - **如果修改该系统调用sys_getdents64()，去掉结果中与那些指定进程相关文件信息，那么就可以达到了隐藏的目的**
      - **int sys_getdents64(unsigned int fd, struct linux_dirent64 *dirp, unsigned int count)**
      - **只需要把上述的sys_getdents64 重载或替换成自己写的hacked_getdents函数，对隐藏的进程文件进行过滤，从而实现进程文件的隐藏。**

7

实验一：Linux进程管理及其扩展

- **(2) 简便方法：将需要隐藏进程的task->pid变成0，就成了0号进程**
  - **在ps，top命令中，是不显示0号进程的相关信息，因此，在/proc/文件夹下就不会有该进程的相关信息了。**

```
asmlinkage int sys_hide( pid_t pid, int on )
{
   struct task_struct *p;
      struct task_struct * me = NULL;
      p = &init_task;
      do
      {          if(p->old_pid ==pid )  // 找到要隐藏或恢复的pid
                 {          me = p;
                            break;
                 }
      }while( ( p = next_task(p) ) && ( p != &init_task ) );
```

8

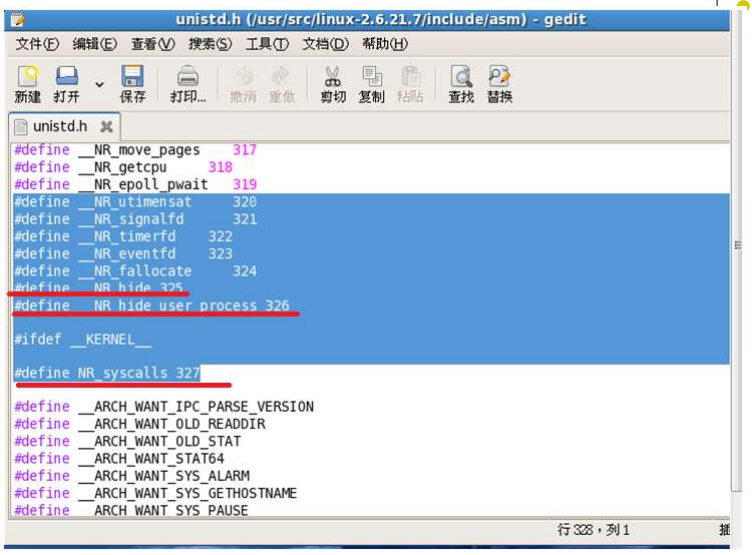
实验一：Linux进程管理及其扩展

```
if( current->uid != 0 || me == NULL ) return 0;
        //判断是否为root用户或者pid是否有效
if( on == 1 )       //执行进程隐藏
{
        me->pid = 0;  //置pid为0，隐藏
        me->hide = 1;  //置隐藏标志为1
}
else                 // 取消进程隐藏
{
        if( me->hide == 1 )
        {
                me->pid = me->old_pid;  //取消隐藏
                me->hide = 0;
        }
}
return 0;
```

9

实验一：Linux进程管理及其扩展

- **在头文件unistd.h中增加自定义的系统调用**



10

实验一：Linux进程管理及其扩展

- 参考资料
  - 1. Linux操作系统实验教程. 北京：电子工业出版社，2009
  - 2．Gray Nutt．Kernel Projects for Linux（影印版）．北京：机械工业出版社,2002
  - 3. Linux进程管理之task_struct结构体 - zxiaocheng - 博客园
    https://www.cnblogs.com/zxc2man/p/6649771.html
  - 4. 浅析task_struct结构体_你又来看我了，一起学习吧-CSDN博客_task_struct
    https://blog.csdn.net/qq_41209741/article/details/82870876

11

实验二： Shell的实现

- 实验目的：
  - 通过实验了解Shell实现机制。
- 实验内容：
  - 实现具有管道、重定向功能的shell，能够执行一些简单的基本命令，如进程执行、列目录等
- 具体要求：
  - 1)设计一个C语言程序，完成最基本的shell角色：给出命令行提示符、能够逐次接受命令；
    对于命令分成三种
    - 内部命令（例如help命令、exit命令等）
    - 外部命令（常见的ls、cp等，以及其他磁盘上的可执行程序HelloWrold等）
    - 无效命令（不是上述二种命令）

12

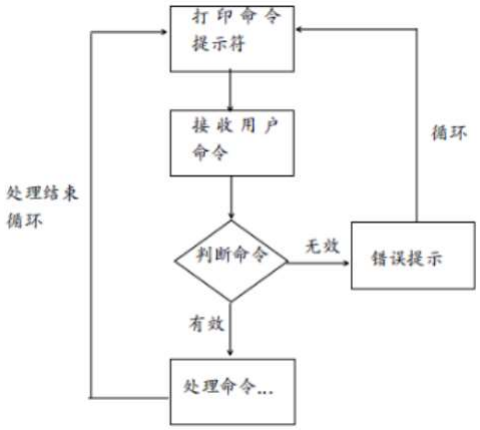
实验二： Shell的实现

- 2)具有支持管道的功能，即在shell中输入诸如“dir | more”能够执行dir命令并将其输出通过管道将其输入传送给more。
- 3)具有支持重定向的功能，即在shell中输入诸如“dir > direct.txt”能够执行dir命令并将结果输出到direct.txt
- 4)将上述步骤直接合并完成

13

实验二： Shell的实现

- **实验指导**
  - **Shell 的基本流程**




```
1  while(TRUE){
2  print_prompt();
3  get_command();
4  if command valid
5  deal_command();
6  else
7      print_error();
8  }
```

14

实验二： Shell的实现

- 重要的几个支撑函数（系统调用）
  - readline()函数
    - 读取用户输入的shell命令，出自开源的库realine（可能需要另外，下载、make）
  - strtok()函数
    - 将一个字符串分解为两个
  - fork()函数
    - 创建一个子进程来执行外部命令，否则会丢失shell
  - Dup()函数
    - 用来“复制”一个打开的文件号，使两个文件号都指向同一个文件
    - linux中的dup()系统调用 - 摩斯电码 - 博客园

    https://www.cnblogs.com/pengdonglin137/p/3286627.html

15

实验二： Shell的实现

- 重要的几个支撑函数（系统调用）
  - Exec函数族（配合fork）
    - exec函数详解_amoscykl的博客-CSDN博客

    https://blog.csdn.net/amoscykl/article/details/80354052

头文件	# include < unistd.h >
函数原型	int execl(const char *path, const char *arg, ...)
	int execv(const char *path, char *const argv[])
	int execle(const char *path, const char *arg, ..., char *const envp[])
	int execve(const char *path, char *const argv[], char *const envp[])
	int execlp(const char *file, const char *arg, ...)
	int execvp(const char *file, char *const argv[])

后缀	含义
l	接收以逗号分隔的参数列表，列表以NULL指针作为结束标志
v	接收到一个以NULL结尾的字符串数组的指针
p	是一个以NULL结尾的字符串数组指针，函数可以通过$PATH变量查找文件

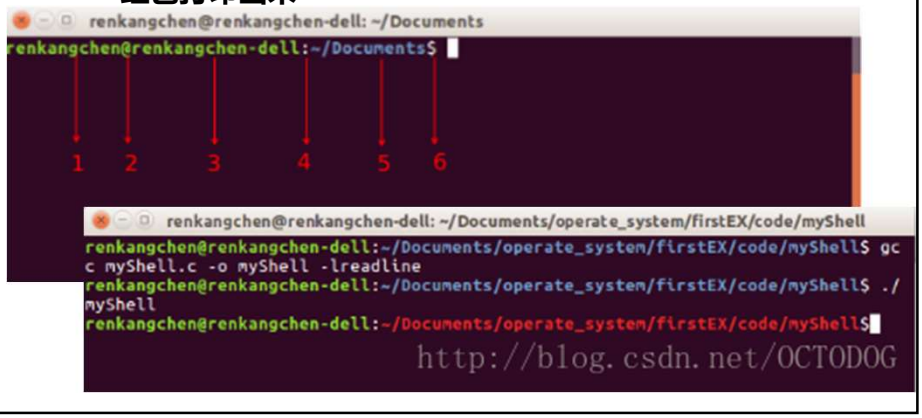
实验二： Shell的实现

- Exec族函数和system的区别
  - system是用shell来调用程序bai=fork+exec+waitpid
  - exec是直接让你的程序代替用来的程序运行，一般和fork联用

17

实验二： Shell的实现

- print_prompt模块 打印命令行提示符
  - 1：用户名； 2： “@”符号； 3：主机名； 4： “：”符号 ； 5：当前目录； 6：用户权限符号(通常均是$,用“#”来表示管理员)
  - 为了避免调试时和原本的shell提示符混淆，新的shell用红色打印出来



实验二： Shell的实现

- get_command模块
  - 用户通常输入命令的一般格式为： 命令 参数选项
    - 通过readline(),我们可以获得用户输入的字符串
    - 使用strtok()函数进行进行分割,这里的分割字符为空格符，将分割后的字符串保存到argv中，这里用argv[0]保存命令名称，argv[1]开始为命令的参数。

```
119         strcpy(argv[0],strtok(command,delims));
120         while(p = strtok(NULL,delims)){
121                 strcpy(argv[i++],p);
122                 argc++;
123         }//while
124
```


19

实验二： Shell的实现

- deal_command模块
  - 内部命令 exit，help，cd，del，move 处理

```
125         //exit when command is exit
126         if(strcmp(argv[0],"exit") == 0){
127                 exit(EXIT_SUCCESS);
128         }
129         else if(strcmp(argv[0],"help") == 0){
130                 help();
131         }//else if
132
```



```
171     else if(strcmp(argv[0],"cd") == 0){
172             char cd_path[100];
173             if((strlen(argv[1])) == 0 ){
174                     pwp = getpwuid(getuid());
175                     sprintf(cd_path,"/home/%s",pwp->pw_name);
176                     strcpy(argv[1],cd_path);
177                     argc++;
178             }
179             else if( !(strcmp(argv[1],"~")) ){
180                     pwp = getpwuid(getuid());
181                     sprintf(cd_path,"/home/%s",pwp->pw_name);
182                     strcpy(argv[1],cd_path);
183             }
184
185             //do cd
186             printf("cdpath = %s \n",argv[1]);
187             if((chdir(argv[1]))< 0){
188                     printf("cd failed!\n");
189             }
190     }//else if cd
```


20

实验二： Shell的实现

- deal_command模块
  - 外部命令
    - 主程序是shell，不能直接利用execve()执行外部命令，否则外部命令替代掉shell，外部命令结束后都主程序也不存在;
    - 需要fork() 子进程去执行外部命令，shell 继续运行，等待下一个shell 命令

```c
#define TRUE 1

while(TRUE) {                   /* repeat forever */
  type_prompt();                /* display prompt on the screen */
  read_command(command,parameters);    /* read input from terminal */
  if(fork()!=0) {               /* fork off child process */
    /* Parent code */
    waitpid(-1,&status,0);          /* wait for child to exit */
   } else {
    /* Child code */
    execve(command,parameters,0);    /* execute command */
  }
}
```


21

实验二： Shell的实现

```
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$ make
gcc  myShell.c -o myShell -lreadline -lncurses
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$ ./myShell
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$ls -l

==>the command is:ls with 2 parameter(s):
@(command): ls
1: -l
@BUILTIN_COMMAND = 0
do command...
argvtmp====
0: ls
1: -l
2: (null)
total 36
-rw-rw-r-- 1 renkangchen renkangchen   134 4月  27 08:55 Makefile
-rwxrwxr-x 1 renkangchen renkangchen 18768 4月  27 23:42 myShell
-rw-rw-r-- 1 renkangchen renkangchen  6089 4月  27 23:42 myShell.c
-rw-rw-r-- 1 renkangchen renkangchen    61 4月  27 18:10 README.md
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$exit

==>the command is:exit with 1 parameter(s):
@(command): exit
@BUILTIN_COMMAND = 1
exit====
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$ git pull
```

1 外部命令

2 外部命令

```
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$ ./
myShell
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$./h
elloworldtest

==>the command is:./helloworldtest with 1 parameter(s):
@(command): ./helloworldtest
hello myShell!
renkangchen@renkangchen-dell:~/Documents/operate_systen/firstEX/code/myShell$
```

3.可执行文件

22

实验二： Shell的实现

- **多重管道的支持**
  - **ls -l | grep fifo | wc –l**

“|”前的表示第一条执行的命令，它的输出结果将通过管道传送给第二条命令，作为第二条命令的参数。

```
$ls -l
总用量 56
-rw-rw-r-- 1 laymond laymond   267 4月   6 18:29 err_seg.c
-rw-rw-r-- 1 laymond laymond   723 4月   7 13:12 fifo_r.c
prw-rw-r-- 1 laymond laymond     0 4月   7 11:07 fifo.temp
-rw-rw-r-- 1 laymond laymond   341 4月   7 10:47 fifo_test.c
-rw-rw-r-- 1 laymond laymond  1425 4月   7 13:12 fifo_w.c
-rwxrwxr-x 1 laymond laymond 10184 4月   6 20:01 fork_n
-rw-rw-r-- 1 laymond laymond   250 4月   6 20:03 makefile
-rwxrwxr-x 1 laymond laymond 10840 4月   8 16:28 pipe_lsgrepwcl_brother
-rw-rw-r-- 1 laymond laymond  2561 4月   8 16:32 pipe_lsgrepwcl_brother.c
-rw-rw-r-- 1 laymond laymond  2190 4月   8 15:47 pipe_lswcl_brother.c
-rw-rw-r-- 1 laymond laymond  1151 4月   8 15:47 pipe_lswcl_fatherandson.c
$ls -l | grep fifo
-rw-rw-r-- 1 laymond laymond   723 4月   7 13:12 fifo_r.c
prw-rw-r-- 1 laymond laymond     0 4月   7 11:07 fifo.temp
-rw-rw-r-- 1 laymond laymond   341 4月   7 10:47 fifo_test.c
-rw-rw-r-- 1 laymond laymond  1425 4月   7 13:12 fifo_w.c
$ls -l | grep fifo | wc -l
4
$
```

23

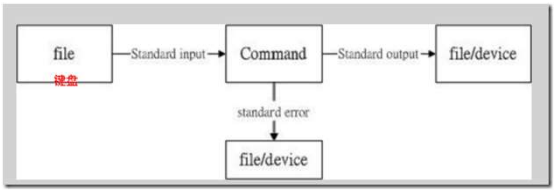
实验二： Shell的实现

- **从本质上说，管道也是一种文件，但它又和一般的文件有所不同：**
  - **无名管道是一个固定大小的缓冲区，linux中大小为1页，即4K字节，因此在写管道时可能变满，当这种情况发生时，随后对管道的write()调用将默认地被阻塞。**
  - **从管道读数据是一次性操作，数据一旦被读，它就从管道中被删除，当所有数据被读走时，管道变空。这时，一个随后的read()调用将默认地被阻塞。**
  - **管道的输出端(读端口) fd[0] 用来给下一个命令/进程提供输入数据**
  - **管道的输入端(写端口) fd[1] 用来容纳上一个命令/进程运行结果**
  - **管道的输入、输出操作是互斥，不能同时进行**

24

实验二： Shell的实现

- 对于任何一条shell命令执行后，会默认打开3个文件描述符:
  - Stdin 标准输入（write端）， 缺省是键盘
  - Stdout 正确输出 （read端），缺省是屏幕
  - Stderr 错误输出，缺省是屏幕



- 这些默认的输出，输入都是linux系统内定的，在使用过程中可以进行重新的设定
  - 命令的输入可以改为文件（包括管道）
  - 命令的输出(正确、错误)可以改为文件（包括管道）

25

实验二： Shell的实现

- 管道实现的核心代码 pipeL(cmd){

```
order = trim(strtok(cmd, "|")); //取得第一个命令
other = trim(strtok(NULL, "")); //取剩下的命令
if (!other)redirect(order); //无管道,执行单个命令
else {                          // 含有管道的多个命令
        pipe(&fd[0]);
        if ((pid = fork()) == 0) { //子进程，执行order并将输出到管道
                close(fd[0]); //管道输入前，先关闭管道输出，互斥
                close(STD_OUT); //关闭标注准输出
                dup(fd[1]); //把标准输出连接到管道输入fd[1]上
                close(fd[1]); //关闭fd[1]
                redirect(order);
        } else { //父进程，递归执行剩下的order
                close(fd[1]);   //管道输出前，先关闭管道输入，互斥
                close(STD_IN);
                dup(fd[0]); //标准输入连接到管道输出 fd[0]上
                close(fd[0]);
                waitpid(pid, &status, 0);
                pipeL(other);   //递归
}}
```

26

实验二： Shell的实现

- **重定向 redirect(order)**
  - **order 可能是**
    - **单条内部命令 如 exit，help，cd，del，**
    - **单条外部命令 如 ls，HelloWorld，需要调用 execve()**
    - **带有重定向的单条命令,需要重定向处理**

```
command                  //type =2
command ">" outfile     //  type =4
command "<" infile     //type =3
command "<" infile ">" outfile | //type = 6
command ">" outfile "<" infile |  //type =5
```

27

实验二： Shell的实现

- **重定向核心代码：**

```
if (type == 4 || type == 5 || type == 6) {
        if ((fd_out = creat(outfile, 0755)) == -1) {
                fprintf(stderr, "#redirect stdout error\n"); return -1;
        }
        close(STD_OUT); dup(fd_out);
        close(fd_out);
  }
  if (type == 3 || type == 5 || type == 6) {
        if ((fd_in = open(infile, O_RDONLY, S_IRUSR | S_IWUSR)) == -1){
                fprintf(stderr, "#can't open inputfile); return -1;
        }
        close(STD_IN);dup(fd_in);
        close(fd_in);
  }
  redirect(order);    (分为内部命令和外部命令)
```

28

实验二： Shell的实现

- 提示：
  - 做这些实验一定要细心，确认无误再编译。等待编译要耐心，一次大概1.5小时左右

29

实验二： Shell的实现

- 参考资料
  - UNIX环境高级编程（第3版）. W. Richard Stevens Stephen A. Rago 著.人民邮电出版社
  - myShell:Linux Shell 的简单实现_山城过雨-CSDN博客_linux myshell dir
    https://blog.csdn.net/OCTODOG/article/details/70942194?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param
  - 用C语言实现简易的shell程序，支持多重管道及重定向_木风feng的博客-CSDN博客_linux myshell 实现多个管道
    https://blog.csdn.net/feng964497595/article/details/80297318?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param
  - Linux简单的shell实现(附源代码) - 百度文库
    https://wenku.baidu.com/view/f6575e5d312b3169a451a486.html

30

实验三：文件系统的实现(选做)

- **实验目的：**
  - **通过实验完整了解文件系统实现机制。**
- **实验内容：**
  - **实现具有设备创建、分区格式化、注册文件系统、文件夹操作、文件操作功能的完整文件系统。**

31

实 验 要 求

- 实验要求由个人独立完成，并在实验现场由助教验收通过。
- 实验需提交实验报告，实验报告应包含如下各项：
  - 基本信息（完成人姓名、学号、报告日期）
  - 实验内容、实验目的、设计思路和流程图
  - 主要数据结构及其说明
  - 源程序并附上注释
  - 程序运行时的初值和运行结果
  - 实验体会

32

机房安排

33