第2章 用户、文件操作与联机帮助

who命令

做什么

知道有谁正在使用系统

进一步了解

- 阅读联机帮助:*man who* → 已登陆用户信息放在/var/run/utmp中(centos7)
- 搜索联机帮助:*man -k utmp* → man 5 utmp(位于第5小节)
- 阅读.h 文件 → more /usr/inclue/utmp.h
  - ut_user保存登录名
  - ut_line保存设备名(用户的终端类型)
  - ut_time保存登陆时间
  - ut_host保存用户用于登陆的远程计算机名字

如何工作

打开utmp
读取记录
显示记录
关闭utmp

如何编写

从文件中读取数据结构

open
- 目标：打开一个文件
- 头文件：#include<fcntl.h>
- 函数原型：int fd = open(char * name, int how)
- 参数
  - name 文件名
  - how O_RDONLY, O_WRONLY, O_RDWR
- 返回值
  - 错误 –1
  - 成功 int → 文件成功打开，内核返回一个正整数的值，就是**文件描述符**
    - 用来唯一标识进程和打开文件之间的连接
    - 必须通过文件描述符对文件操作
    - 同时打开多个文件，文件描述符不同
    - 一个文件打开多次，文件描述符也不同

read
- 目标：把数据读取到缓冲区
- 头文件：#include<unstd.h>
- 函数原型：ssize_t numread = read(int fd, void * buf, size_t qty)
- 参数
  - fd 文件描述符
  - buf 用来存放数据的目的缓冲区
  - qty 要读取的字节数
- 返回值
  - 错误 –1
  - 成功 numread

close
- 目标：
- 头文件：#Include<unistd.h>
- 函数原型：int result = close(int fd)
- 参数：fd 文件描述符
- 返回值
  - 错误 –1
  - 成功 0

who1.c待改进
- 消除空白记录 → ut_type != USER_PROSESS 直接return
- 正确显示登陆时间 → print("% 12.12s", ctime(&t) + 4)

cp命令

做什么

cp source-file targer-file

如何创建/重写文件

create
- 目标：创建/重写一个文件
- 头文件：#include<fcntl.h>
- 函数原型：int fd = create(char * filename, mode_t mode)
- 参数
  - filename 文件名
  - mode 访问模式
- 返回值
  - 错误 –1
  - 成功 fd

write
- 目标：将内存中的数据写入文件
- 头文件：#include<unistd.h>
- 函数原型：ssize_t result = write(int fd, void * bud, size_t amt)
- 参数
  - fd 文件描述符
  - Buf 内存数据
  - amt 要写的字节数 → 实际写入字节数小于要求的情况
    - 系统对文件最大尺寸限制
    - 磁盘空间接近满
- 返回值
  - 错误 –1
  - 成功 num wirtten

如何编写

打开源文件准备读取
打开目标文件准备写
从源文件读取数据写入缓冲区 → EOF
将缓冲区数据写入目标文件
关闭源文件
关闭目标文件

使用缓冲提高文件I/O效率

缓冲区的大小对性能有很大的影响, 原因是系统调用需要在用户–内核之间转换, 会消耗很多时间
原来的who命令每次只取一个用户, 大大浪费了时间, 可以一次取16条记录

内核缓冲技术

内核也使用缓冲技术来提高对磁盘的访问速度, 磁盘是数据块的集合, 内核会对磁盘上的数据块做缓冲
内核将磁盘的数据块复制到内核缓冲区中, 当一个用户空间中的进程要从磁盘上读数据时, 内核一般不直接读磁盘, 而是将内核缓冲区中的数据复制到进程的缓冲区中
当进程所要求的数据块不在内核缓冲区时, 内核会把相应的数据块加入到请求数据列表中, 然后把该进程挂起, 接着为其他进程服务
当内核把相应的数据读到内核缓冲区, 然后再把数据复制到进程缓冲区中, 最后唤醒被挂起的进程

导致的结果
- 提高磁盘I/O效率
- 优化磁盘的写操作
- 需要及时地将缓冲数据写入磁盘

文件读写

系统调用每次打开一个文件都会保存一个指向文件位置的指针, 当读写操作完成时, 指针会移动到下一个记录位置, 这个指针与文件描述符相关联.这种情况下, 我们如何修改文件中的内容. 答案是系统调用lseek
指针是与文件描述符项关联, 而不是与文件关联, 所以如果两个程序同时打开一个文件, 这时会有两个指针, 两个程序对文件的读操作不会互相干扰

lseek

目标：使指针指向文件中的指定位置
头文件： #include <sys/type.h> 或者 #include <unistd.h>
函数原型：off_t oldpos = lseek(int fd, off_t dist, int base)
参数
- fd 文件描述符
- dist 移动的距离
- base
  - SEEK_SET 文件开始
  - SEEK_CUR 当前位置
  - SEEK_END 文件结尾

返回值
- 错误 –1
- 指针变化前的位置 oldpos

一些例子
- 把指针往前偏移一个 utmp 结构：lseek(fd, –(sizeof(struct utmp)), SEEK_CUR);
- 把指针指到第11个记录的开始位置：lseek(fd, 10 * sizeof(struct utmp), SEEK_SET);
- 把指针只到文件末尾：lseek(fd, 0, SEEK_END);
- 返回指针所指向的当前位置：lseek(fd, 0, SEEK_CUR);

终端注销代码

```
logout_tty(char *line)
{
    int fd;
    struct utmp rec;
    int len = sizeof(struct utmp);
    int retval = -1;

    if ( ( fd = open(UTMP_FILE, O_RDWR)) == -1)
        return -1;

    while (read(fd, &rec, len) == len)
        if (strncmp(rec.ut_line, line, sizeof(rec.ut_line)) == 0)
        {
            rec.ut_type = DEAD_PROCESS;
            if (time(&rec.ut_time != -1))
                if (lseek(fd, -len, SEEK_CUR) != -1)
                    if (write(fd, &rec, len) == len)
                        retval = 0;
            break;
        }
    if (close(fd) == -1)
        retval = -1;
    return retval;
}
```

上面函数接受一个line参数, 用于对比 struct utmp 中的 ut_line 成员, 来确定当前记录, 读到当前记录后, 更改属性, 并使用 lseek 将文件指针回退一个 utmp 长度, 并将新的 utmp 写入到文件

处理系统调用中的错误

确定错误种类: errno

```
#define    EPERM    1      /* Operation not permitted */
#define    ENOENT   2      /* No such file or directory */
#define    ESRCH    3      /* No such process */
#define    EINTR    4      /* Interupted system call */
#define    EIO      5      /* I/O error */
```

不同错误需要不同的处理
显示错误信息: perror(3)