Linux操作系统编程

Linux文件系统概述

creat(建立文件)

- **头文件**

  #include<sys/types.h>
  #include<sys/stat.h>
  #include<fcntl.h>

- **函数原型**

  int creat(const char * pathname, mode_t mode)

- **参数说明**

参数pathname指向欲建立的文件路径字符串。creat()相当于使用下列的调用方式调用open()。
open(const char * pathname ,(O_CREAT|O_WRONLY|O_TRUNC))

- **返回值**

  creat()会返回新的文件描述词，若有错误发生则会返回-1，并把错误代码设给errno

- **附加说明**

  creat函数的一个不足之处是它以只写方式打开所创建的文件

close(关闭文件)

- **头文件**
  #include<unistd.h>
- **函数原型**
  int close(int fd);
- **函数说明**
  当使用完文件后若已不再需要则可使用close()关闭该文件，close()会让数据写回磁盘，并释放该文件所占用的资源。参数fd为先前由open()或creat()所返回的文件描述符;
- **返回值**
  若文件顺利关闭则返回0，发生错误时返回-1。
- **附加说明**
  虽然在进程结束时，系统会自动关闭已打开的文件，但仍建议自行关闭文件，并检查返回值。

read(由已打开的文件读取数据)

- **头文件**

  #include<unistd.h>

- **定义函数**

  ssize_t read(int fd,void * buf ,size_t count);

- **函数说明**

  read()会把参数fd 所指的文件传送count个字节到buf指针所指的内存中。若参数count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是无可读取的数据，此外文件读写位置会随读取到的字节移动;

- **返回值**

  成功返回读取的字节数，出错返回-1。

read函数

- **read函数实际读到的字节数少于要求读的字节数时：**
  - 读普通文件，在读到要求字节数之前就到达文件尾；
  - 当从终端设备读，通常一次最多读一行；
  - 当从网络读时，网络中的缓冲机构可能造成返回值小于所要求读的字节数；
  - 某些面向记录的设备，如磁带，一次最多返回一个记录；
- **读操作完成后，文件的当前位置将从读之前的位置加上实际读的字节数**
- **当有错误发生时则返回-1，错误代码存入errno中，而文件读写位置则无法预期**

write(将数据写入已打开的文件内)

- **头文件**
  #include<unistd.h>
- **函数原型**
  ssize_t write (int fd,const void * buf,size_t count);
- **函数说明**
  write()会把参数buf所指的内存写入count个字节到参数fd所指的文件内。当然，文件读写位置也会随之移动;
- **返回值**
  如果顺利write()会返回实际写入的字节数。当有错误发生时则返回-1，错误代码存入errno中。write出错的原因可能是磁盘满、没有访问权限、或写超过文件长度限制等等
- **附加说明**
  将数据写入已打开的文件内。对于普通文件，写操作从文件当前位置开始写（除非打开文件时指定了O_APPEND选项）。写操作完成后，文件的当前位置将从写之前的位置加上实际写的字节数

write函数

- **数据无法一次性读完时**
  - 第二次读buf中数据时，读位置指针并不会自动移动
  - 按如下格式实现读位置移动：write(fp, p1+len, (strlen(p1)-len)，直至指针恢复
- **Write一次可以写的最大数据范围是8192**
  - 写入数据大小最好小于buff中的值
  - Count参数值大于SSIZE_MAX，则write调用的结果未定义
  - Count参数值为0时，write调用会立即返回0这个值
- **Write调用返回时，内核已经将缓冲区所提供的数据复制到内核的缓冲区，但是无法保证数据已经写出到预定的目的地**

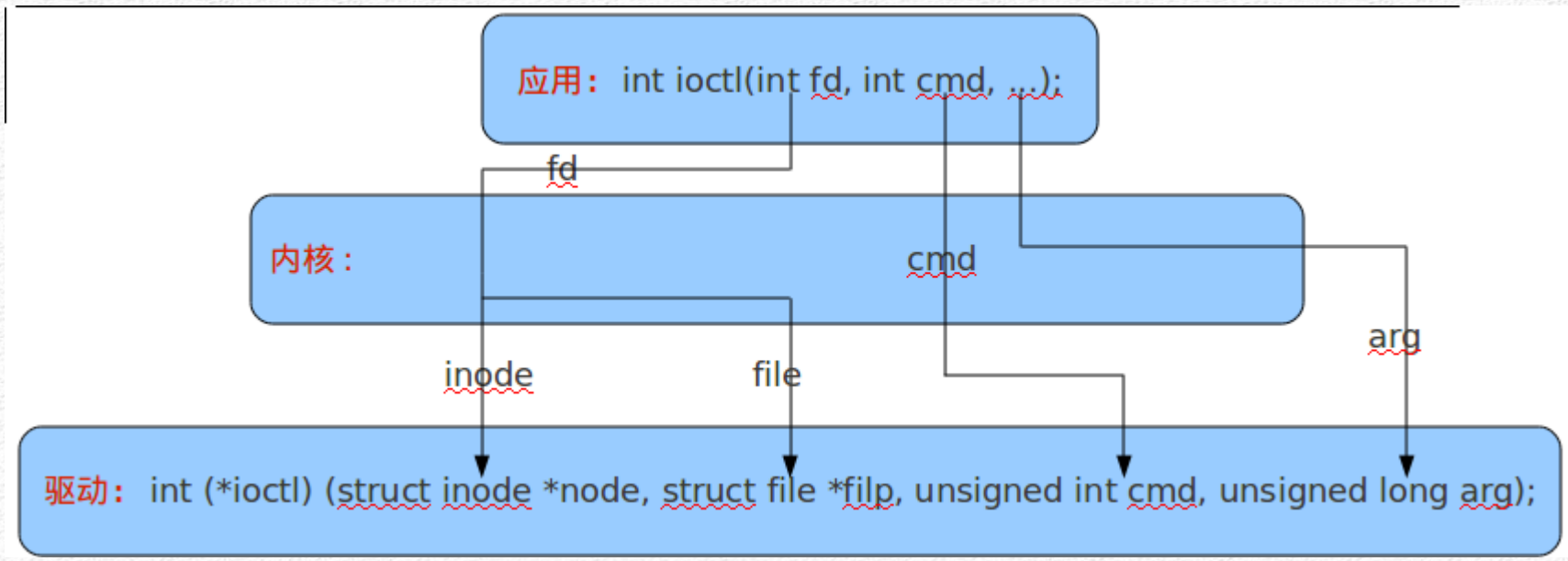
Read和write函数示例

```
#include <stdio.h>
#include <stdlib.h>
 #include <unistd.h>
 int main(void)
{
      char buf[100];
      int num = 0;
      if ((num = read(STDIN_FILENO, buf, 10)) == -1)
      {
               printf ("read error"); error(-1);
      } else {
              // 将键盘输入又输出到屏幕上
               write(STDOUT_FILENO, buf, num);
      }
return 0;
 }
```

ioctl(设备驱动程序中对设备的I/O通道进行管理)

- **头文件**
  #include<sys/ioctl.h>
- **定义函数**
  int ioctl(int fd, int cmd, ...);
- **函数说明**
  ioctl()能对一些特殊的文件(主要是设备)进行一些底层参数的操作。许多字符设备都使用ioctl请求来完成对设备的控制;
- **返回值**
  成动返回0。当有错误发生时则返回-1，错误代码存入errno中
- **附加说明**
  ioctl是设备驱动程序中对设备的I/O通道进行管理的函数。所谓对I/O通道进行管理，就是对设备的一些特性进行控制，例如串口的传输波特率、马达的转速等等。

Ioctl函数



应用层与驱动函数的ioctl之间的联系

ioctl函数

- 在驱动程序中实现的ioctl函数体内，实际上是有一个switch{case}结构，每一个case对应一个命令码，做出一些相应的操作
- ioctl中命令码是唯一联系用户程序命令和驱动程序支持的途径

设备类型（幻数）	序列号	方向	数据尺寸
8bit	8bit	2bit	8~14bit

- "幻数"是一个字母，数据长度也是8，用一个特定的字母来标明设备类型