文件 IO 编程

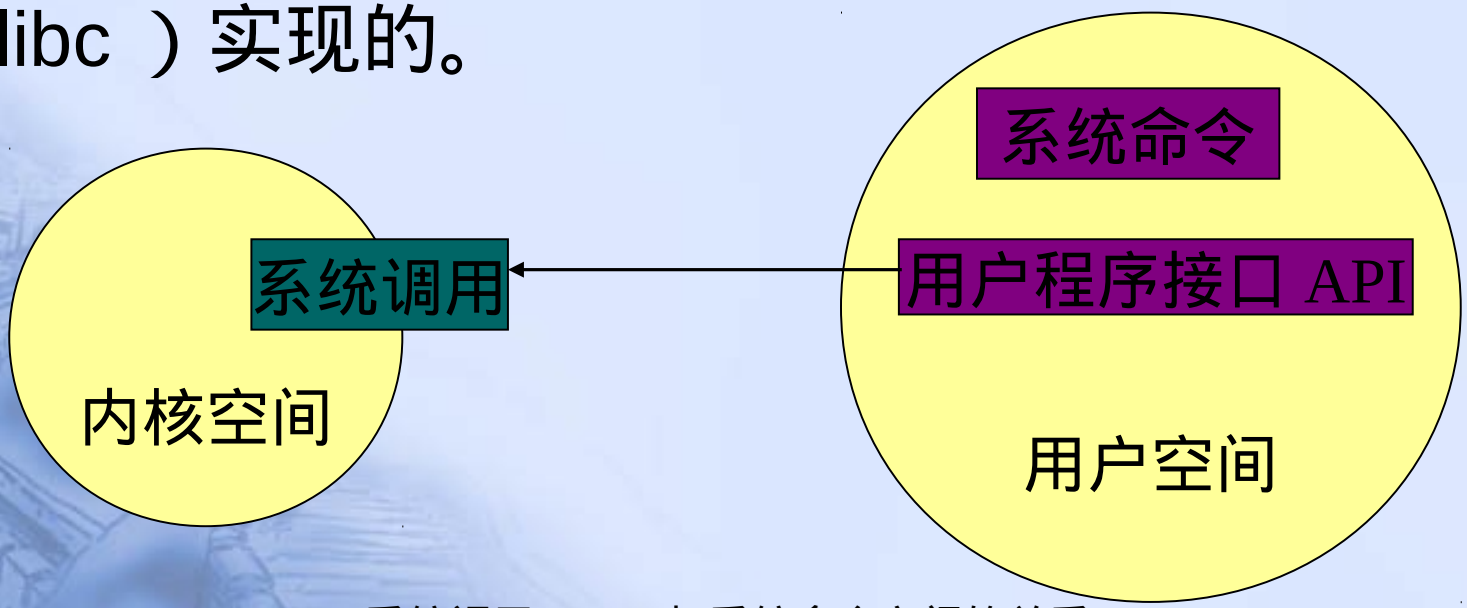
嵌入式资源免费下载
http://www.kontronn.com

Linux 系统调用与文件 I/O

- Linux 系统调用 (system call)
- 所谓系统调用是指操作系统提供给用户程序的一组“特殊”接口，用户程序可以通过这组“特殊”接口来获得操作系统内核提供的特殊服务。
- 在 Linux 中用户程序不能直接访问内核提供的服务。为了更好的保护内核空间，将程序的运行空间分为内核空间和用户空间，他们运行在不同的级别上，在逻辑上是相互隔离的。

用户程序接口（ API ）

- 在 Linux 中用户编程接口（ API ）遵循了在 UNIX 中最流行的应用编程界面标准— POSIX 标准。这些系统调用编程接口主要通过 C 库（ libc ）实现的。



系统调用、 API 与系统命令之间的关系

文件 I/O 介绍

- 可用的文件 I / O 函数——打开文件、读文件、写文件等等。大多数 Linux 文件 I / O 只需用到 5 个函数：
  open 、 read 、 write 、 lseek 以及 close 。
- 不带缓存指的是每个 read 和 write 都调用内核中的一个系统调用。这些不带缓存的 I / O 函数不是 ANSI C 的组成部分，而是 POSIX 组成部分。

文件描述符

- 对于内核而言，所有打开文件都由文件描述符引用。文件描述符是一个非负整数。当打开一个现存文件或创建一个新文件时，内核向进程返回一个文件描述符。当读、写一个文件时，用 open 或 creat 返回的文件描述符标识该文件，将其作为参数传送给 read 或 write 。

文件描述符

- 在 POSIX.1 应用程序中，整数 0、 1、 2 应被代换成符号常数：
  - STDIN_FILENO
  - STDOUT_FILENO
  - STDERR_FILENO
- 这些常数都定义在头文件 <unistd.h> 中。
- 文件描述符的范围是 0 ~ OPEN_MAX 。早期的 UNIX 版本采用的上限值是 19 (允许每个进程打开 20 个文件)，现在很多系统则将其增加至 256。

open 函数

- #include <sys/types.h>
- #include <sys/stat.h>
- #include <fcntl.h>
- int open(const char *pathname, int oflag, …/*, mode_t mode * /) ;
- 功能：打开文件
- 返回：若成功为文件描述符，若出错为 - 1

open 函数

- pathname 是要打开或创建的文件的名字。
- oflag 参数可用来说明此函数的多个选择项。
- 对于 open 函数而言，仅当创建新文件时才使用第三个参数。
- 用下列一个或多个常数进行或运算构成 oflag 参数 (这些常数定义在 <fcntl.h> 头文件中) ：
  - O_RDONLY 只读打开。
  - O_WRONLY 只写打开。
  - O_RDWR 读、写打开。

- O_APPEND 每次写时都加到文件的尾端。
- O_CREAT 若此文件不存在则创建它。使用此选择项时，需同时说明第三个参数 mode ，用其说明该新文件的存取许可权位。
- O_EXCL 如果同时指定了 O_CREAT ，而文件已经存在，则出错。这可测试一个文件是否存在，如果不存在则创建此文件成为一个原子操作。
- O_TRUNC 如果此文件存在，而且为只读或只写成功打开，则将其长度截短为 0 。
- O_NOCTTY 如果 pathname 指的是终端设备，则不将此设备分配作为此进程的控制终端。
- O_NONBLOCK 如果 pathname 指的是一个 FIFO 、一个块特殊文件或一个字符特殊文件，则此选择项为此文件的本次打开操作和后续的 I / O 操作设置非阻塞方式。
- O_SYNC 使每次 write 都等到物理 I / O 操作完成。

creat 函数

- #include <sys/types.h>
- #include <sys/stat.h>
- #include <fcntl.h>
- int creat(const char * pathname, mode_t mode) ;
- 功能：创建一个新的文件。
- 返回：若成功为只写打开的文件描述符，若出错为 - 1 。
  - 注意，此函数等效于：
  - open (pathname, O_WRONLY | O_CREAT | O_TRUNC, mode) ;
  - creat 的一个不足之处是它以只写方式打开所创建的文件。

close 函数

- #include <unistd.h>
- int close (int *filedes*) ;
- 功能：关闭一个打开文件
- 返回：若成功为 0 ，若出错为 - 1
- 　　当一个进程终止时，它所有的打开文件都由内核自动关闭。很多程序都使用这一功能而不显式地用 close 关闭打开的文件。

read 函数

- #include <unistd.h>
- ssize_t read(int fd, void *buf, size_t count);
- 功能： 从打开文件中读数据
- 返回：读到的字节数，若已到文件尾为 0 ，若出错为 - 1 。
- 注意： count 是期望读取的最大字节数，而 read 的返回值是实际读取到的字节数。一般情况下，二者会相等，但当文件中的剩余数据不足 count 时，则二者不会相同，这一般发生在最后一次读取

read 函数

- 有多种情况可使实际读到的字节数少于要求读字节数：
  - 读普通文件时，在读到要求字节数之前已到达了文件尾端。例如，若在到达文件尾端之前还有 30 个字节，而要求读 100 个字节，则 read 返回 30，下一次再调用 read 时，它将返回 0 (文件尾端) 。
  - 当从终端设备读时，通常以行为单位，读到换行符就返回。
  - 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，例如磁带，一次最多返回一个记录。
- 读操作从文件的当前位移量处开始，在成功返回之前，该位移量增加实际读得的字节数。

write 函数

- #include <unistd.h>
- ssize_t write(int fd, const void *buf, size_t count);
- 功能：向打开文件写数据。
- 返回：若成功为已写的字节数，若出错为 - 1 。
- 　　其返回值通常与参数 *count* 的值相同，返回 -1 表示出错。write 出错的一个常见原因是：磁盘已写满，或者超过了对一个给定进程的文件长度限制。

- 对于普通文件，写操作从文件的当前位移量处开始。如果在打开该文件时，指定了 O_APPEND 选择项，则在每次写操作之前，将文件位移量设置在文件的当前结尾处。在一次成功写之后，该文件位移量增加实际写的字节数。

lseek 函数

- 每个打开文件都有一个与其相关联的“当前文件偏移量”。它是一个非负整数，用以度量从文件开始处计算的字节数。通常，读、写操作都从当前文件偏移量处开始，并使偏移量增加所读或写的字节数。按系统默认，当打开一个文件时，除非指定 O_APPEND 选择项，否则该位移量被设置为 0 。
- 可以调用 lseek 显式地定位一个打开文件的读写指针到指定位置。

lseek 函数

- #include <sys/types.h>
- #include <unistd.h>
- off_t lseek(int filesdes, off_t offset, int whence) ;
- 功能：设置文件内容读写位置
- 返回：若成功为新的文件位移，若出错为 - 1 。
- 对参数 offset 的解释与参数 whence 的值有关。
  - 若 whence 是 SEEK_SET ，则将该文件的位移量设置为距文件开始处 offset 个字节。
  - 若 whence 是 SEEK_CUR ，则将该文件的位移量设置为其当前值加 offset， offset 可为正或负。
  - 若 whence 是 SEEK_END ，则将该文件的位移量设置为文件长度加 offset， offset 可为正或负。

文件属性

- #include <sys/types.h>
- #include <sys/stat.h>
- #include <unistd.h>
- int stat(const char *path, struct stat *buf);
- 功能：查看文件或目录属性
- 返回值：成功返回 0，错误返回 -1

- stat() 函数用来将参数 path 所指的文件状态复制到参数 buf 所指的结构中 .

```
struct stat {
        dev_t     st_dev;       /* ID of device containing file */
        ino_t     st_ino;       /* inode number */
        mode_t    st_mode;      /* protection */
        nlink_t   st_nlink;     /* number of hard links */
        uid_t     st_uid;       /* user ID of owner */
        gid_t     st_gid;       /* group ID of owner */
        dev_t     st_rdev;      /* device ID (if special file) */
        off_t     st_size;      /* total size, in bytes */
        blksize_t st_blksize; /* blocksize for filesystem I/O */
        blkcnt_t  st_blocks;  /* number of blocks allocated */
        time_t    st_atime;   /* time of last access */
        time_t    st_mtime;   /* time of last modification */
        time_t    st_ctime;   /* time of last status change */
    };
```

- int fstat(const int filedes, struct stat *buf);
- int lstat(const char *pathname, struct stat *buf);

文件类型

宏	文件类型
S_ISREG()	普通文件
S_ISDIR()	目录文件
S_ISCHR()	字符特殊文件
S_ISBLK()	块特殊文件
S_ISFIFO()	管道或FIFO
S_ISLNK()	符号连接（POSIX.1或SVR4无此类型）
S_ISSOCK()	套接字（POSIX.1或SVR4无此类型）

权限部分的实现（权限位宏）

- S_ISUID　　执行时设置 - 用户 - ID
- S_ISGID　　执行时设置 - 组 - ID
- S_ISVTX　　保存正文

- S_IRWXU　　用户（所有者）读、写和执行
- S_IRUSR　　用户（所有者）读
- S_IWUSR　　用户（所有者）写
- S_IXUSR　　用户（所有者）执行

- S_IRWXG　　组读、写和执行
- S_IRGRP　　组读
- S_IWGRP　　组写
- S_IXGRP　　组执行

- S_IRWXO　　其他读、写和执行
- S_IROTH　　其他读
- S_IWOTH　　其他写
- S_IXOTH　　其他执行

目录操作

- 打开目录： opendir
  - DIR *opendir(const char *name);
- 读取目录： readdir
  - struct dirent *readdir(DIR *dir);
- 关闭目录： closedir
  - int closedir(DIR *dir);

打开目录

- #include <sys/types.h>
- #include <dirent.h>
- DIR *opendir(const char *name);
- 功能： opendir() 用来打开参数 name 指定的目录，并返回 DIR* 形态的目录流。
- 返回值：成功返回目录流，失败返回 NULL

读取目录

- #include <sys/types.h>
- #include <dirent.h>
- struct dirent *readdir(DIR *dir);
- 功能： readdir() 返回参数 dir 目录流下的一个子条目（子文件或子目录）。
- 返回值：成功返回结构体指针，错误或已读完目录返回 NULL 。
  - 结构说明：
  - struct dirent {
  - ino_t d_ino; /* inode number */
  - off_t d_off; /* offset to the next dirent */
  - unsigned short d_reclen; /* length of this record */
  - unsigned char d_type; /* type of file */
  - char d_name[256]; /* filename */
  - };

关闭目录

- #include <sys/types.h>
- #include <dirent.h>
- int closedir(DIR *dir);
- 功能： closedir() 关闭 dir 所指的目录流。
- 返回值：成功返回 0，失败返回 -1, 错误原因存在 errno 中。