

《嵌入式工程师自修养》视频教程

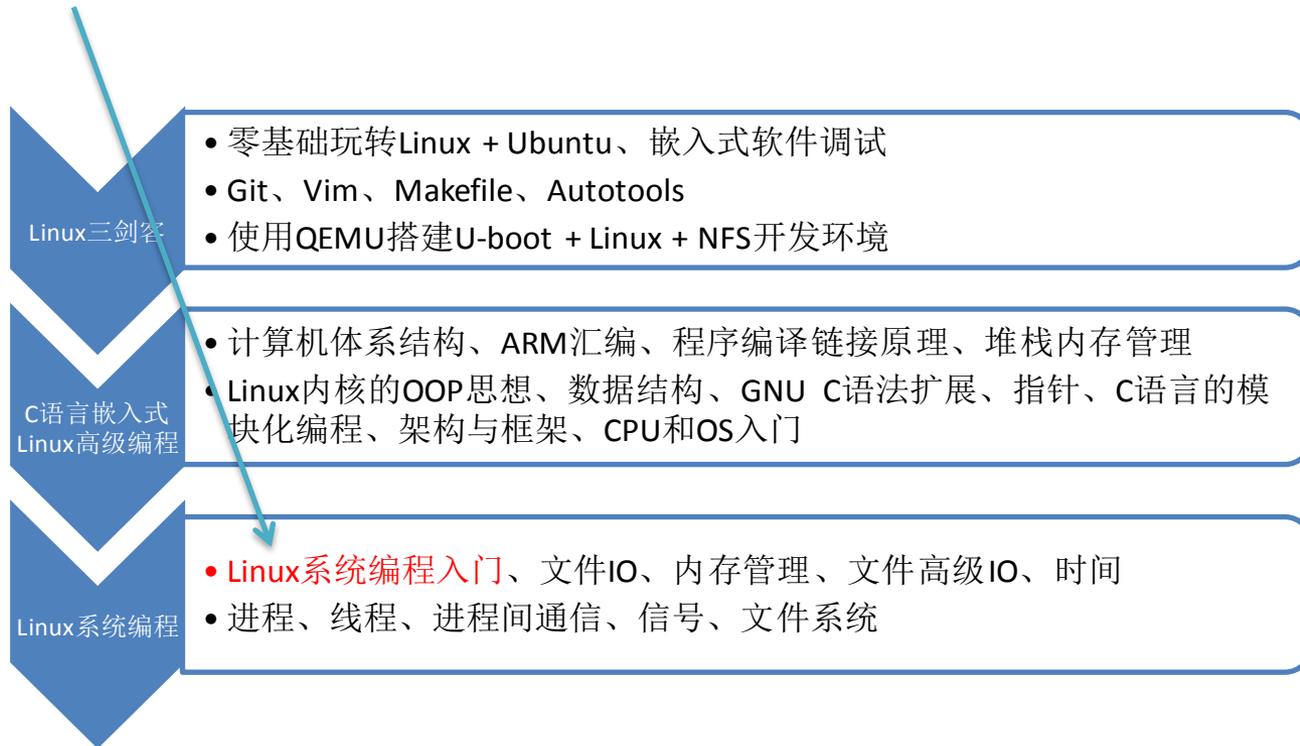
- 详情咨询QQ: 3284757626
- 视频淘宝店: wanglitao.taobao.com
- 博客: www.zhaixue.cc
- 微信公众号:



Linux系统编程：入门篇

Roadmap

- We are here...



什么是系统编程？

- 代码复用
 - 函数
 - 库
 - 框架
 - 操作系统

什么是系统编程？

- 使用uC/OS的API创建任务

- 调用OS相关功能
- 调用系统相关功能
- 调用硬件相关功能

```
void task(void *pd)
{
    ;
}
OS_STK stack[1024];
int main()
{
    Bsplnit();
    OSInit();
    OSTaskCreate(task,(void *)0,&stack[1023],1);
    OSStart();
}
```

Linux系统编程

- 系统编程范畴
 - 内核态系统编程：Linux内核编程
 - 用户态系统编程：系统调用
- 系统调用
 - 将留给应用程序的内核API接口统一管理
 - 硬件访问权限、特权指令
 - Linux系统调用接口
 - 文件IO、目录、文件系统
 - 进程、线程、进程间通信
 - 时间、信号、网络
 - 内存管理

学习Linux系统编程有什么用？

- 职业发展

- 武功技能：内功+招式
- 系统软件工程师：系统编程+具体业务
 - 互联网
 - 嵌入式
 - 系统管理
 - 银行金融
 - 驱动开发

一个系统编程例子(上)： 文件基本操作

系统调用

- 文件I/O

- `int open(const char *pathname, int flags, mode_t mode);`
- `int close(int fd);`
- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, const void *buf, size_t count);`

- 头文件

- `sys/stat.h`
- `sys/types.h`
- `fcntl.h`
- `unistd.h`

系统调用函数

- 文件的打开、关闭

- `int open(const char *pathname, int flags, mode_t mode);`

```
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
int main(void)
{
    int fd;
    fd = open("hello.txt",O_RDONLY|O_CREAT,0666);
    if( fd == -1){
        printf("open file failed!\n");
        return -1;
    }
    close(fd);
    return 0;
}
```

参数选项

Flags(主参数)	说明
O_RDONLY	以只读方式打开文件
O_WRONLY	以只写方式打开文件
O_RDWR	以可读可写方式打开
Flags2(副参数)	说明
O_CREAT	若文件不存在，则创建该文件
O_EXCL	若使用O_CREAT且文件存在，返回错误
O_TRUNC	若文件已经存在则删除文件中的数据
O_APPEND	以追加方式打开文件
O_SYNC	同步方式打开文件，刷新缓冲区到文件
O_NONBLOCK	以非阻塞方式打开文件
O_ASYNC	异步I/O方式读写文件
args3(文件权限)	说明
mode	8进制权限码，0766：文件owner、用户组、其它用户权限

系统调用函数

- 文件的读写

- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, const void *buf, size_t count);`
- `off_t lseek(int fd, off_t offset, int whence);`
- `fsync(int fd);`

系统调用函数

- 文件定位
 - lseek(int fd, off_t offset, int whence);
 - SEEK_SET: 文件开始
 - SEEK_CUR: 文件当前位置
 - SEEK_END: 文件末尾

实现cp命令

实现cp命令

- **基本流程**

- 打开源文件
- 打开目标文件
- 读取源文件数据
- 写入到目标文件

思考

- 文件相关的read、write、close系统调用函数都在unistd.h中
- 为什么open函数要单独放在fcntl.h文件中？

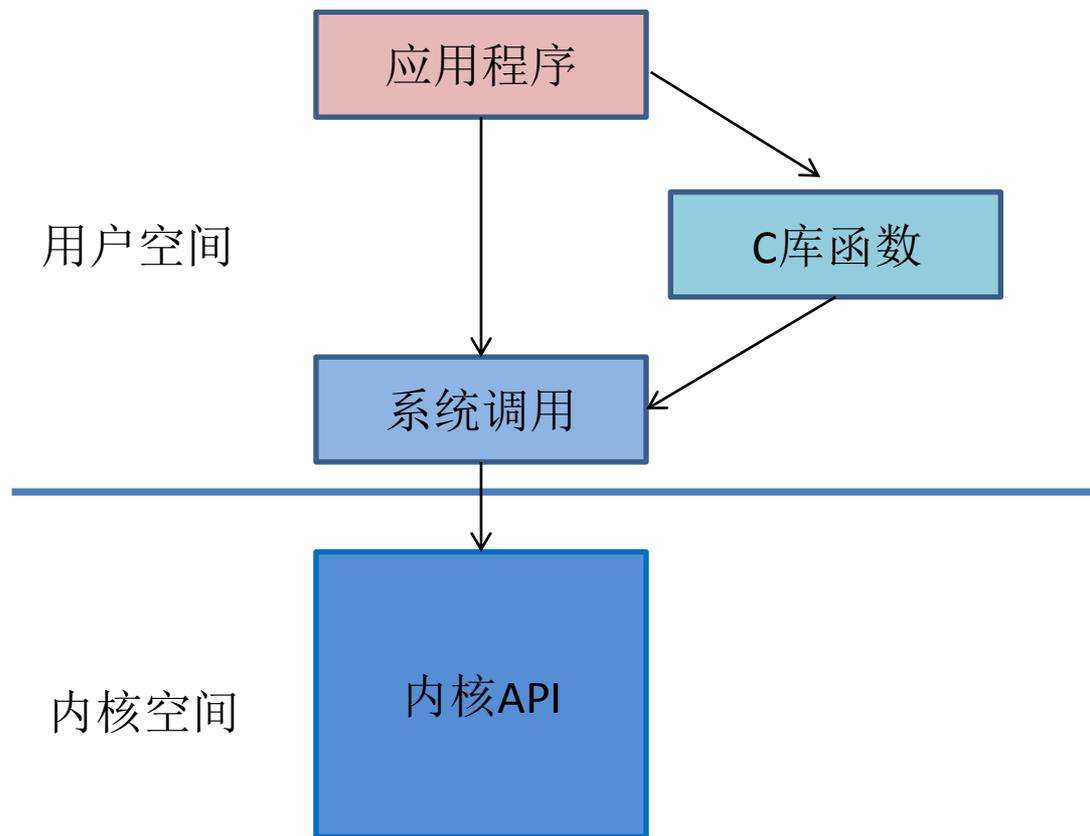
系统调用与C标准库

C标准库

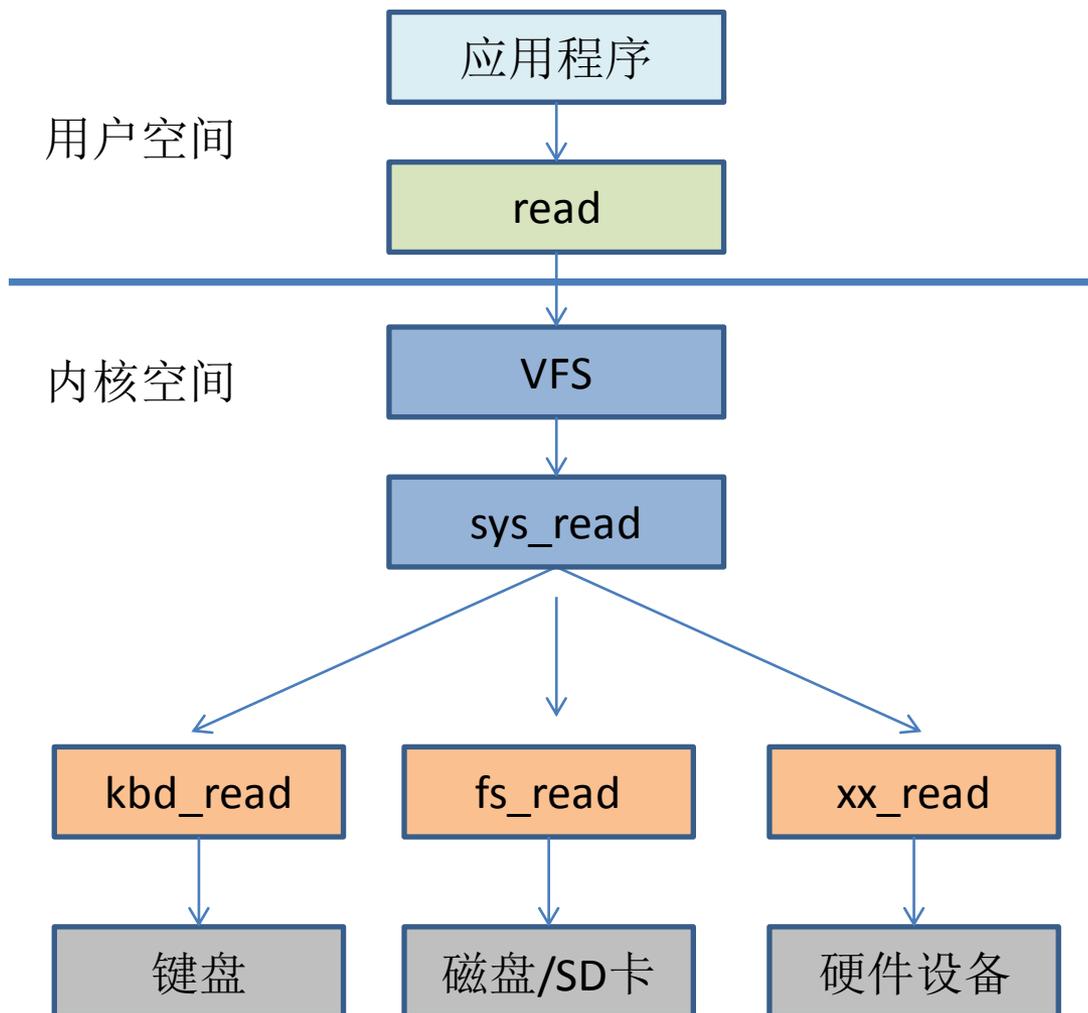
- 文件操作

- `#include <stdio.h>`
- `FILE *fopen (const char *path, const char *mode);`
- `int fclose (FILE *stream);`
- `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `int fseek (FILE *stream, long offset, int whence);`
- `long ftell (FILE *stream);`
- `void rewind (FILE *stream);`

系统调用与C标准库函数



Linux系统调用



Linux系统调用

- **基本流程**

- 软中断：X86下int 0x80；ARM架构下 SWI软中断指令
- 寄存器保存相关参数：参数、系统调用号
- 进入内核态，执行内核特权指令代码函数
- 返回值保存到寄存器
- 返回到用户态、系统调用结束

思考

- 既然有了系统调用API，为什么还要使用C标准库函数？

思考

- 系统调用的API哪里来的？
- 对应的头文件(如unistd.h)哪里来的？

POSIX标准

问题

- 系统调用API
 - 不同OS之间的差异
 - 相同OS，不同版本、分支之间的差异

POSIX标准

- POSIX简介

- 可移植操作系统接口：Portable Operating System Interface of UNIX
 - 系统调用API
 - Shell命令
- POSIX并不局限于UNIX
 - Linux/ Ubuntu / Fedora / Debian
 - FreeBSD
 - Solaris
 - Windows
 - Mac OS X / IOS
 - Sun OS

标准与兼容

- 标准的发展
 - SVID: system V interface definition
 - SUS: single Unix Specification
 - POSIX: POSIX.1
 - SUSV4: 合并了POSIX.1标准

Linux系统调用

- 常见的系统调用
 - 文件IO
 - 进程、线程
 - 进程间通信
 - 系统控制
 - 内存管理
 - 网络管理

API

- API
 - ABI-Application Programming Interface
 - 应用程序和库之间的编程接口
 - POSIX、C标准

ABI

- ABI
- ABI-Application Binary Interface
- 字节序、数据类型、大小和对齐
- 调用约定：参数传递、寄存器使用，如ATPCS
- 系统调用编码
- 链接、库行为
- 目标文件、可执行文件的二进制格式
- 一个可执行文件可以在支持ABI标准的不同平台上运行

man命令

man命令

- man手册

- NAME: 命令名称
- SYNOPSIS: 头文件及函数原型
- CONFIGURETION: 介绍说明
- DESCRIPTION: 函数介绍
- OPTIONS: 可选参数选项
- EXIT STATUS: 出错返回结果
- RETURN VALUE: 返回值
- ERRORS: 出错返回值
- ENVIRONMENT
- FILES: 用到的文件
- VERSIONS
- CONFORMING TO
- NOTES: 注意事项
- BUGS
- EXAMPLE: 使用范例
- AUTHORS

命令手册

- 格式: `man -n 命令`
 - 1: 普通应用程序或shell命令
 - 2: 系统调用
 - 3: 库函数
 - 4: 特殊文件(通常是指设备文件)
 - 5: 文件格式或协议(如/etc/passwd文件各个字段的含义)
 - 6: 游戏程序
 - 7: 混杂设备
 - 8: 系统管理命令(针对root用户)
 - 9: 非标准的内核程序

TIPS

- 在源代码中查看帮助
 - vim: shift + k
 - vim基本操作：翻页、查找、下一个

info命令

info文档

- 与man命令相比

- 存储位置: /usr/share/info
- Info页面编写得更加友好、易于理解
- Info页面内容由多个区段（节点）组成，翻页显示(man滚屏显示)
- 超文本链接

info命令

- 基本命令

- info + 命令
- N: 显示下一个节点的页面内容
- P: 显示上一个节点的页面内容
- M: 先敲M进入命令行，输入命令名查看该命令帮助文档
- L: 返回上一个访问节点的页面内容
- SPACE: 向前滚动一页
- BACK/DEL: 向后滚动一页
- B/E: 一个节点内容的开始/结束
- H: 打开info教程
- D: 回到info的初始节点
- Enter: 跳转到链接文本
- Q: 退出

strace命令

strace命令

- **基本功能**

- 监控用户进程与内核进程的交互
- 追踪进程的系统调用、信号传递、状态变化

系统调用

- 分类

- 文件和设备访问：open、close、read、write、ioctl等
- 进程管理：fork、clone、execve、exit等
- 信号：signal、kill等
- 内存管理：brk、mmap、mlock等
- 进程间通信：semget、信号量、消息队列
- 网络通信：socket、connect等

strace使用

- 常用参数

- -c: Count time, calls, and errors for each system call
- -d: Show some debugging output of strace itself on the standard error
- -f/F: Trace child processes created by currently traced processes
- -h: Print the help summary
- -k: Print the execution stack trace of the traced processes
- -r: Print a relative timestamp upon entry to each system call
- -t: Prefix each line of the trace with the time of day
- -tt: Prefix each line of the trace with the time of day
- -T: Show the time spent in system calls

strace使用

- 常用参数

- -e set: only trace some system calls
- -e open,close: only trace open/close system calls
- -e file: only trace file system calls
- -e process: Trace all system calls which involve process management
- -e network: Trace all the network related system calls
- -e signal : Trace all signal related system calls.
- -e ipc: Trace all IPC related system calls.
- -e desc: Trace all file descriptor related system calls.
- -e memory: Trace all memory mapping related system calls.
- -e set: Trace only the specified subset of signals.
- -o filename: Write the trace output to the file filename
- -p pid: Trace the process with the process ID pid

strace使用

- 使用示例

- 查看一个程序所有的open、close系统调用
- 查看每个系统调用消耗的时间
- 统计系统调用次数、错误次数统计
- 打印系统调用的时间戳
- 将跟踪日志保存到log文件中

错误处理

错误处理

- 系统调用

- 系统编程错误一般通过函数的返回值表示
- 执行成功，返回0或正确值
- 执行失败，返回-1，并把系统全局变量errno赋值，指示具体错误
- 全局变量errno由操作系统维护：当系统调用或调动库函数出错时，会重置该值

errno值

- 定义

- /usr/include/asm-generic/errno.h
- /usr/include/asm-generic/errno-base.h

GNU编码规范

编程规范

- 编程是一门艺术
 - GNU 编程标准
 - Linux内核编码风格
 - XX公司编码规范
 - XX实验室编程规范
 - 自成风格?

编程规范

- 代码风格

- 变量函数命名、空白、注释、文档
- 安全性：减少bug、增强稳定性
- 可读性：易于理解
- 维护性：易于管理、维护
- 拒绝代码风格：python

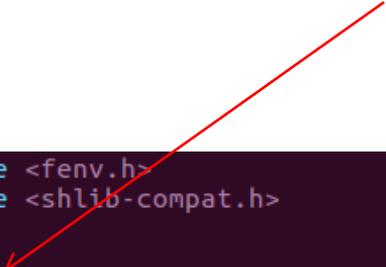
国际C语言混乱代码大赛

2018 almyr	Converts text to sound using font as spectrogram
2018 anderson	Visualizer of typographic rivers
2018 bellard	Image compression demo
2018 burton1	Hex dumper
2018 burton2	Tokenize and count
2018 ciura	Strunk & White checker
2018 endoh1	Animated GIF from text
2018 endoh2	Monty-Pythonesque animated quine
2018 ferguson	Dawkins' weasel simulator
2018 giles	SDL falling sand
2018 hou	Converter of JSON to SVG pie chart
2018 mills	PDP-7/11 simulator
2018 poikola	Ursa Major ASCII animation
2018 vokes	Computing strongly connected graph components
2018 yang	Text rotator and shifter

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef unsigned
long W;
static
W*i,t,c,h,e,r,y,**a,*b,*o,*u,*n,*d,*s;enum
_ {F=6 , I=7,N= 5 , D = 3 , M=14
,Y=0 , C=8,L = 1 , U = 9 , E=4};
static void (* _ ) ( ) ; static void*(*(B)()),*T;
static char m[0x3543],*Z;void *meltndown(void *p,size_t
sz){ void *z=realloc(p,sz);if(!z||sz)exit(n?y!=""n:I-F);
return z;}void magic(W*p){W j;for(j=0;j<p[F];j++){W*v=T,Q=
p[M+j];W*J=a[Q];if(N[J]==~0LU){ _ (Q);v=&I[J];}else if(J[D])
{v=&N[J];}if(v&&I[p]>*v)I[p]=*v;}void cast(void){B(i,Y);}void
spell(W G,W H){ _ ( ); _ (H); _ (H, G); _ (H); _ (G);}void witch(
void){e=n[ --E[b]];a[e][D]=0;if(b[C]>=U[b]){c=2*(b[C+L]?b[U]:1);o=B
(d,c *sizeof*o);u=B(s,c*sizeof*o);b[M-N]=c;d=o;s=u;}d[b[M+d]]+=e;if
(e>y)y=e;}void*brew(void*g,size_t l){puts(l?"":"Out of range.");exit
(g?1:1);}void newt(W *p){F[p]=0;N[p]=I[p]==F[p];}void bubble(W H){for
(t=0;t<H;t++){s[i[d[H-t-1]]--1]=d[H-t-1];}void boil(W G){for(t=0;t<
G-1;t++){i[t+1]=i[t];} _ =bubble;}void hex(W G,W H){for(t=0;t<H?G:Y;t++
){i[d[t]]++;} _ =boil;}void nasal_demons(W G){for (t=Y;t<G;t++)i[t]=0; _
=hex;}void toil(void){i=B(i,y*sizeof*i); _ =nasal_demons;}void bat(W l)
{C[b]=0; _ =witch;do _ ();while(e!=1);y++;if(!y)B=brew; _ =toil;spell (C
[b],y);printf("%lu:" ,b[L]);for(y=0;y<C[b];y++){printf(" %lu" ,s[y]);}y=
0;puts("");}void potion(W l){W*p=a[l];if(p[N]!=~0LU)return;p[F+L]=h;p[N]
=h;p[D]=1;h++;n[b[L+D]]+=1;if(E[b]==r){r*=2;n=B(n,r*sizeof*n);}magic(p)
;if(I[p]==p[N]){bat(l);} _ =potion;}void O(void){n=B(n,2*sizeof*n);Y[n
]=r;if(a&&a[t]){r=2;b=a[t]; _ =potion;b[L+C]=b[L]^b[L];E[b]=1;for(y=0;y
<*n;y++){if(a[y]){ _ (y);}} _ =cast;h=0;}void toad(W g,W j,W*l){o=B(a
[g],((a[g]?a[g][F]:0)+j*M)*sizeof*o); _ =newt;if(!a[g]){ _ (o);}for(y=
0;y<j;y++){o[F[o]+F+C]=l[y];o[F]++;if(l[y]==g){continue;}u=B(a[l[y]]
,((a[l[y]?a[l[y]][F]:Y)+M)*sizeof*o);if(!a[l[y]]){ _ (u);a[l[y]]
=u;}y=0;a[g]=o;}void familiar(void){W**w=e;r;if(!r){r=1;while
(r<c-t){r*=2;if(!r){B=brew;break;}}w=B(a,r*sizeof*a);if(!
Z){Z=" ";for(y=e;y<r;y++){y[w]=T;}y=0;a=w;
_ (Y[i] ,c-1,&i[1]);}void spectre(void ){if((W
*_Z-060
>011){
return;}
_ =toad
; i[c]=(
_ w)atoi(
Z);if(i[c]>t){t=i[c];c++;if(c==h){i=B(i,2*h*sizeof*i);h*=2;}y=i[c-1];if
((Z=strotok(T," ")){if(U=(W)*Z-I*I+L) _ =spectre;}}int main(int argv,
char**argc){h++;if(argc){T=argc[argv];} _ =0;i=B(meltdown)(T,h*sizeof
*i);while(*argc||Z){c=0;*argc=fgets(m,sizeof(m),stdin); _ =spectre;if(!*
argc){Z=T;goto _ ;}Z=strotok(*argc," ");while(Z&&(W)*Z-(M*D+F)<=U){ _ (
);}if(t>r)familiar();else if(c>0){ _ (i[Y],c-1,&i[1]);} _ ();goto _ ;}
```


GNU编码风格：函数顶头写

```
20 #include <fenv.h>
21 #include <shlib-compat.h>
22
23 int
24 __fesetenv (const fenv_t *envp)
25 {
26     #if defined FE_NOMASK_ENV && FE_ALL_EXCEPT != 0
27         if (envp == FE_NOMASK_ENV)
28             return 1;
29     #endif
30     /* Nothing to do. */
31     return 0;
32 }
```



GNU编码风格：空格

- 使用空格的地方

- 左括号之前
- 逗号之后
- 运算符两边

```
#include <sys/wait.h>
#include <sys/resource.h>
#include <math-tests.h>
#define ALL_EXC \
    (INEXACT_EXC | DIVBYZERO_EXC | UNDERFLOW_EXC | OVERFLOW_EXC | \
     INVALID_EXC)
/* Test whether a given exception was raised. */
static void
test_single_exception (short int exception,
                      short int exc_flag,
                      fexcept_t fe_flag,
                      const char *flag_name)
{
    if (exception & exc_flag)
    {
        if (fetestexcept (fe_flag))
            printf (" Pass: Exception \"%s\" is set\n", flag_name);
        else
        {
            printf (" Fail: Exception \"%s\" is not set\n", flag_name);
            ++count_errors;
        }
    }
    else
    {
        if (fetestexcept (fe_flag))
        {
            printf (" Fail: Exception \"%s\" is set\n", flag_name);
            ++count_errors;
        }
        else
        {
            printf (" Pass: Exception \"%s\" is not set\n", flag_name);
        }
    }
}
```

GNU编码风格：对齐

- 多行对齐

```
#include <sys/wait.h>
#include <sys/resource.h>
#include <math-tests.h>
#define ALL_EXC \
    (INEXACT_EXC | DIVBYZERO_EXC | UNDERFLOW_EXC | OVERFLOW_EXC | \
     INVALID_EXC)
/* Test whether a given exception was raised. */
static void
test_single_exception (short int exception,
                      short int exc_flag,
                      fexcept_t fe_flag,
                      const char *flag_name)
{
    if (exception & exc_flag)
    {
        if (fetestexcept (fe_flag))
            printf (" Pass: Exception \"%s\" is set\n", flag_name);
        else
        {
            printf (" Fail: Exception \"%s\" is not set\n", flag_name);
            ++count_errors;
        }
    }
    else
    {
        if (fetestexcept (fe_flag))
        {
            printf (" Fail: Exception \"%s\" is set\n", flag_name);
            ++count_errors;
        }
        else
        {
            printf (" Pass: Exception \"%s\" is not set\n", flag_name);
        }
    }
}
```

GNU编码风格：命名

- 命名风格

- 函数名、变量名用小写字母、下划线分割
- 宏、枚举常量使用大写字母定义
- 函数命名： `open_door`
- Windows风格： `OpenDoor`

Linux哲学：一切皆文件

Linux哲学

- 设计思想

- 一个程序只实现一个功能，多个程序组合完成复杂功能
- 数据配置在文本文件中
- 机制与策略
- 一些皆文件

一切皆文件

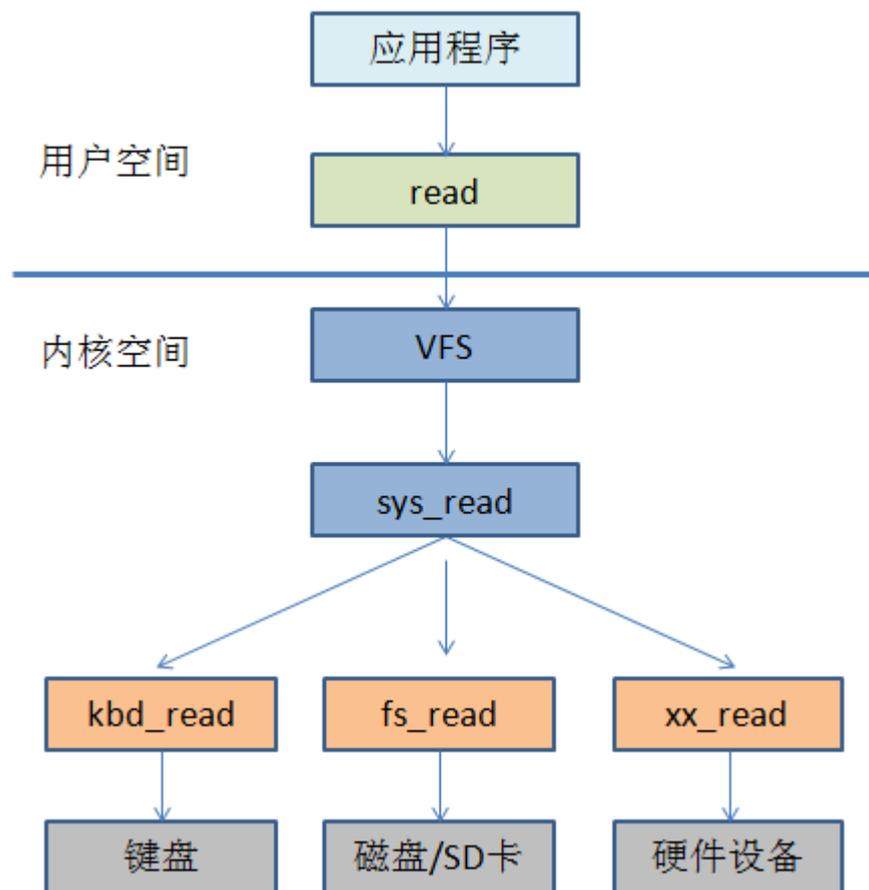
- 广义上的文件

- 普通文件
- 目录
- 特殊设备文件：字符设备、块设备、网络设备
- 套接字
- 进程、线程
- 命名管道

VFS

- 虚拟文件系统

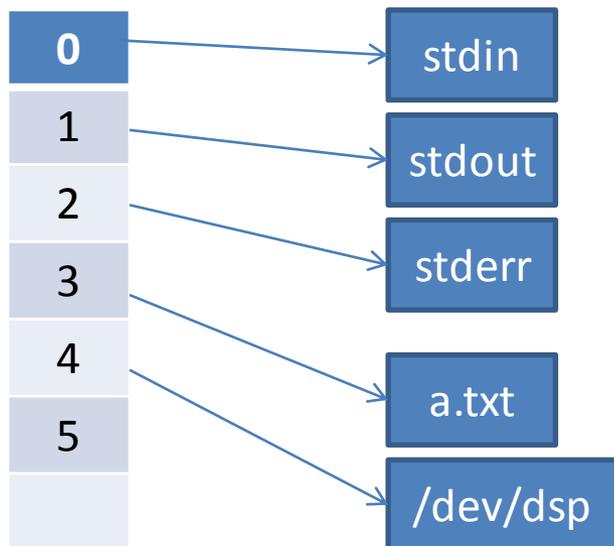
- visual file system
- 对不同文件系统进行抽象
- 提供一个统一的操作接口



文件描述符

- 基本概念

- file descriptor简称fd，用来描述一个已经打开的文件索引(非负整数)
- 当fd为负时，表示文件打开失败



音频播放器(上)

设备文件

- 设备文件

- 普通文件可以通过文件名与实际的存储数据进行关联
- 设备文件通过设备节点与具体的物理设备进行关联
- 设备号：主设备号 + 次设备号组成
- 设备文件存在于/dev目录下
- 设备节点可以自动创建、也可以手工创建

常见的设备文件

设备文件	说明	设备节点	说明
null	空设备	cdrom	光盘设备
port	I/O端口	fb0	帧缓冲设备
mem	物理内存镜像	loop	循环设备
kmem	虚拟内存镜像	psaux	鼠标
aio	异步I/O	zero	零流源设备
kmsg	printk缓冲区	sda	SCSI磁盘
console	控制台	random	随机数发生器
tty	当前tty设备	vcs	虚拟控制台内容
ttyS	串口	dsp	音频输出设备
snd/	声卡设备	mixer	混音控制器

音频播放器

- 设备节点dsp
 - 打开声卡设备文件/dev/dsp
 - 打开音频数据文件xx.wav
 - 从音频文件读一段数据
 - 写入声卡设备文件
 - 关闭文件

音频播放器(下)

声卡参数

• 采样率

- 取样频率：22050(立体声)、44.1(CD音质)、48(超级CD音质)、192K
- 采样频率越高音质越好
- 不同声卡支持的采样率不同，软件设置要根据硬件进行设置

• 采样格式

- 量化精度：16bit、20bit、24bit
- PCM：线性脉冲编码调制
- 文件格式：CD(高音质)、WAV(无损)、MP3(压缩)、MIDI、WMA
- wav文件：44.1K采样率、16bit量化位数

• 声道设置

- 声音在录制或回放时在不同空间位置采集的相互独立的音频信号
- 声道数(channels)：声音录制时的音源数量或播放时的扬声器数量
- 分类：单声道、双声道立体声、四声道、5.1杜比环绕、7.1声道

ioctl 系统调用

- 基本使用

- 头文件:
- 原型: `int ioctl (int fd, unsigned long int cmd, ...);`
- 用途:
 - 向设备发送控制参数、配置命令
 - 获取设备信息

后续课程学习路线

后续课程学习路线

- Linux系统编程
- Linux内核编程

Linux系统编程

- **基础技能**

- 文件IO、文件和目录、缓存、
- 内存管理、信号
- 进程、线程、进程间通信

- **项目提升**

- 网络编程
- GUI编程
- 嵌入式项目开发

Linux内核编程

- **基础技能**

- 模块机制、系统调用原理、进程
- 中断、内核同步、内存管理
- 定时器、文件系统、VFS

- **项目提升**

- 驱动开发
- 内核开发
- 内核项目实战