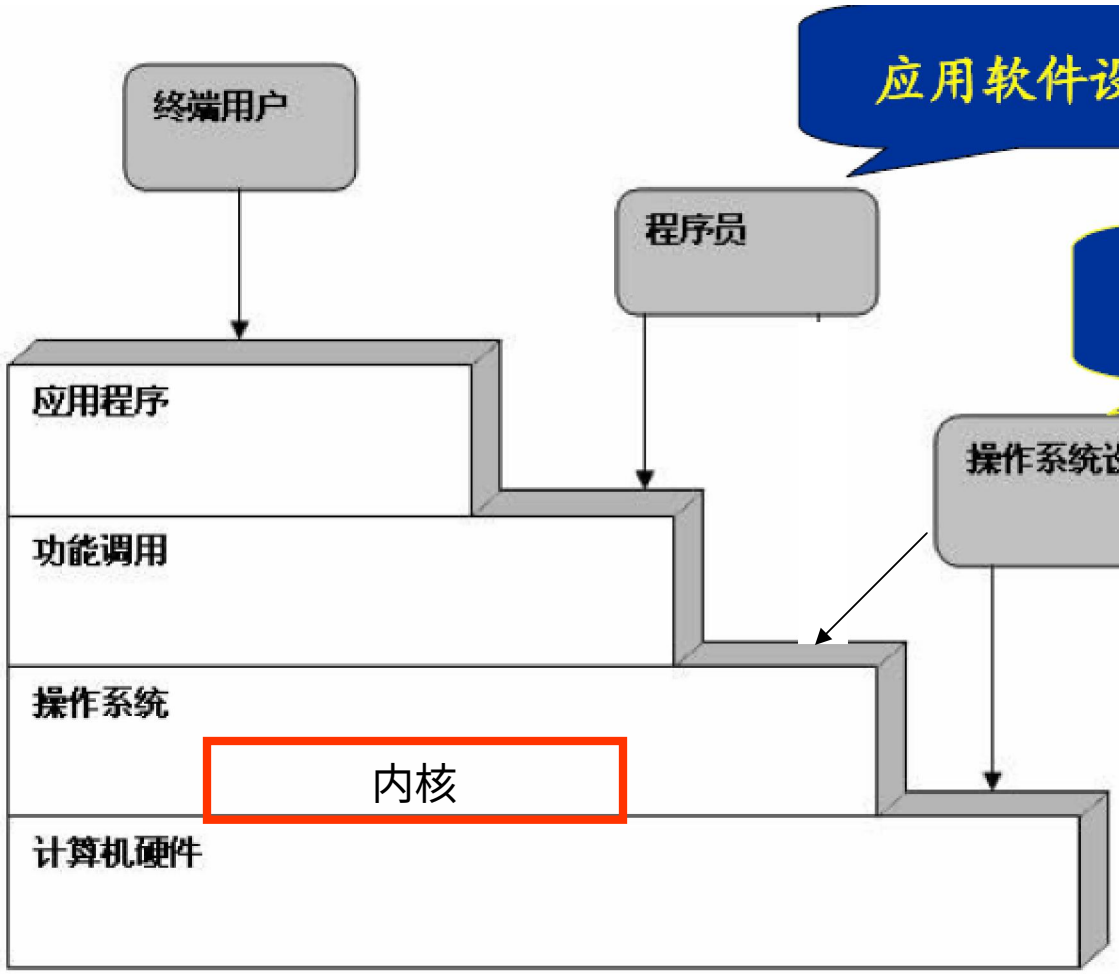
0、概述

——Linux内核原理

为什么要学？
如何学好
Linux 内核原理、设备驱动和嵌入式系统开发

软件开发者的层次划分

高级语言，开发工具，软件工程, etc.



…，低级语言，理解OS，编译原理,硬件体系结构（X86，ARM，MIPS，PPC，M86K，etc.）

如何学好

- C的基础
- 理解OS原理
- 了解硬件体系结构（相应的汇编基础）
- 了解编译原理

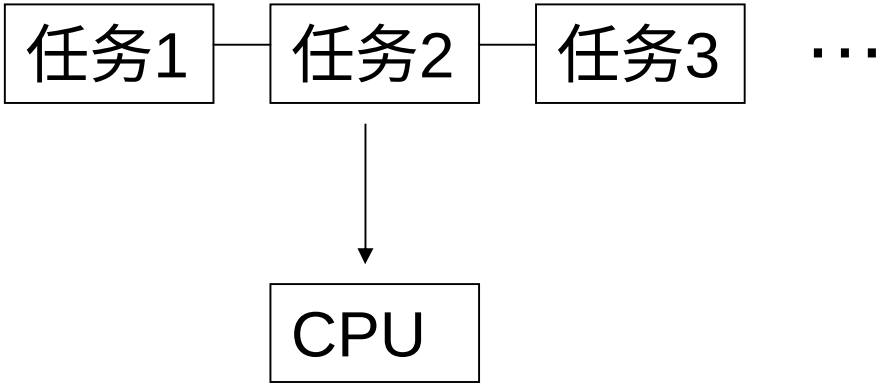
示例：内存管理

```
#include <stdio.h>
#define SIZE 1024*4
int main()
{
        char *p=0;
        p=malloc(SIZE);
        p[SIZE-1]='a';
        free(p);
        printf("p=%c\n",p[SIZE-1]);
        p[SIZE-1]='b';
        printf("p=%c\n",p[SIZE-1]);

        return 0;
        }
```

SIZE=1024*128 ?

示例：定时器精度

- setitimer
- 时间片
- 任务负载



示例：代码效率

```
// 第 一 个
for (i=0; i<N; i++)
{
if (condition)
    DoSomething();
else
    DoOtherthing();
}
```

```
// 第 二 个
if (condition)
{
for (i=0; i<N; i++)
    DoSomething();
}
else
{
    for (i=0; i<N; i++)
    DoOtherthing();
}
```

示例：（i386）

```
struct animal_struct {
        char dog;               /* 1字节*/
        unsigned long cat;      /* 4字节*/
        unsigned short pig;     /* 2字节*/
        char fox;               /* 1字节*/
};
```

```
struct animal _struct {
        char dog;               /* 1字节*/
        u8 __pad0[3];           /* 3字节*/
        unsigned long cat;      /* 4字节*/
        unsigned short pig;     /* 2字节*/
        char fox;               /* 1字节*/
        u8 __pad1;              /* 1字节*/
};
```

```
struct animal_struct {
        unsigned long cat;      /* 4字节*/
        unsigned short pig;     /* 2字节 */
        char dog;               /* 1字节*/
        char fox;               /* 1字节*/
};
```

sizeof(anim_struct)=?

实例：/include/linux/kernel.h

```
#define min(x,y) ({ \
        typeof(x) _x = (x); \
        typeof(y) _y = (y); \
        (void) (&_x == &_y); \
        _x < _y ? _x : _y; })
```

```
#define container_of(ptr, type, member) ({                  \
        const typeof( ((type *)0)->member ) *__mptr = (ptr); \
        (type *)( (char *)__mptr - offsetof(type,member) );})
```

了解编译原理

- 预处理(Pre-Processing)
  - 调用预处理程序cpp，展开在源文件中定义的宏“#include”头文件；
- 编译(Compiling)
  - 调用ccl将c文件编译成汇编文件.S；
- 汇编(Assembling)
  - 调用as将汇编代码编译成目标代码.o；
- 链接(Linking)
  - 用链接程序ld,把生成的目标代码链接成一个可执行程序a.out，elf，coff。

函数调用

- 函数调用时，调用者依次把参数压栈，然后调用函数
- 函数被调用以后，在堆栈中取得数据，并进行计算。
- 函数计算结束以后，或者调用者、或者函数本身修改堆栈，使堆栈恢复原装。

示例：

```
void c()
{
     printf("In func. c.\n");
     }
void b(unsigned int * b)
{
     b[3]=0x08048354;
     printf("In func. b.\n");
     }
void a(int q)
{
     unsigned int a[2]={3,5};
     printf("In func. a.\n");
     b(a);
     }
int main()
{
     int q=1;
     a(q);
     return 0;
     }
```

结果：

In func. a

In func. b

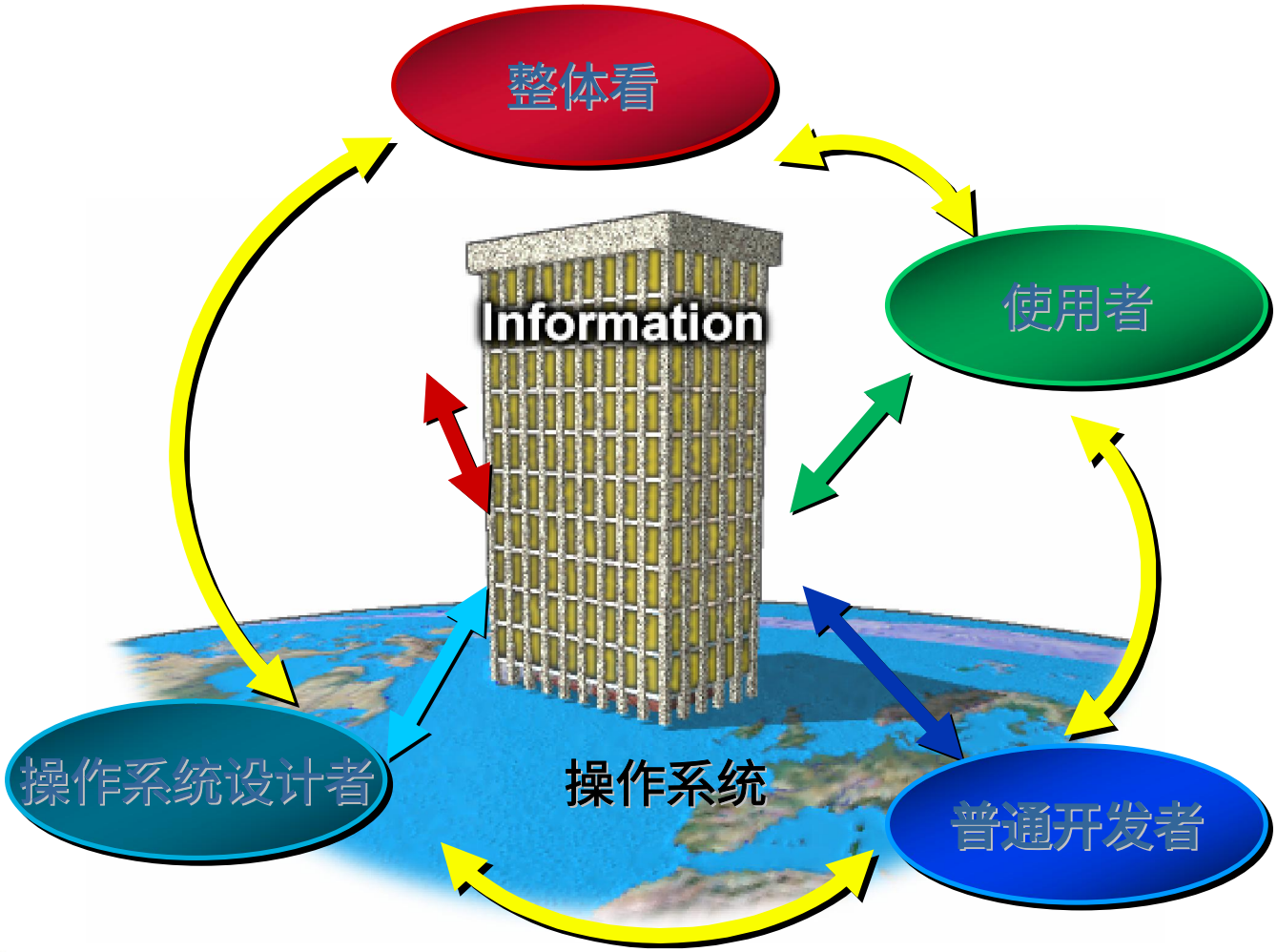
In func. c

示例:

```
#include <stdio.h>
void (*ttt)(void) ;
char pfunc[]={0xe8,0xcf,0xed,0xff,0xff,0xc3};

void entry()
{
        printf("Here I am!\n");
}

int main()
{
    ttt=(void *) pfunc;
    ttt();

    return 0;
    }
```

OS原理

不同角度看到的操作系统



认识操作系统－从使用者的角度看

- 要拷贝一个文件，具体的拷贝操作是谁完成的？
  - 你需要知道文件存放在何处吗？
  - 柱面、磁道、扇区描述什么？
  - 数据的搬动过程怎样进行
- 繁琐留给自己，简单留给用户
  - cp /home/test1 /home/test2

认识操作系统－从应用开发者的角度看

- cp的实现

```
inf=open("/floppy/TEST",O_RDONLY,0);
out=open("/mydir/test",O_WRONLY,0600);
  do{
l=read(inf,buf,4096);
write(outf,buf,l);
} while(l);
close(outf);
close(inf);
```

认识操作系统－从OS的角度看

- 要拷贝一个文件，具体的拷贝操作
  - 通过文件系统将路径转化为磁头、柱面、扇区、块号。
  - 通过磁盘驱动的中断服务例程读写磁盘
  - 利用缓冲区对内核和cp进程的地址空间进行数据交换。

OS概念

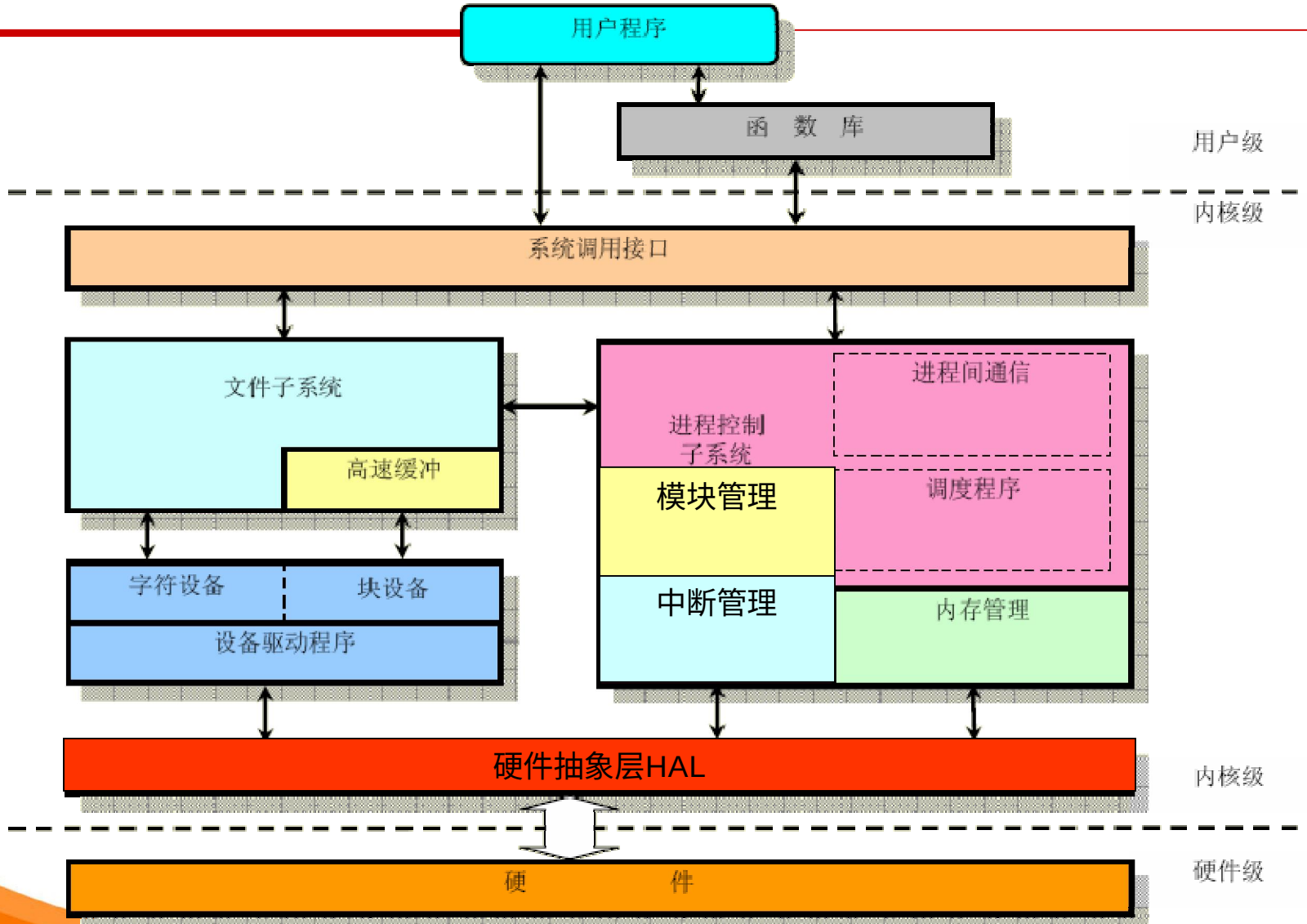
操作系统是计算机系统中的一个系统软件，是一些程序模块的集合——它们能以尽量有效、合理的方式组织和管理计算机的软硬件资源，合理的组织计算机的工作流程，控制程序的执行并向用户提供各种服务功能，使得用户能够灵活、方便、有效的使用计算机，使整个计算机系统能高效、顺畅地运行。

操作系统的主要目标

- 方便用户使用
- 管理系统资源
- 提高系统效率
- 构筑开放环境

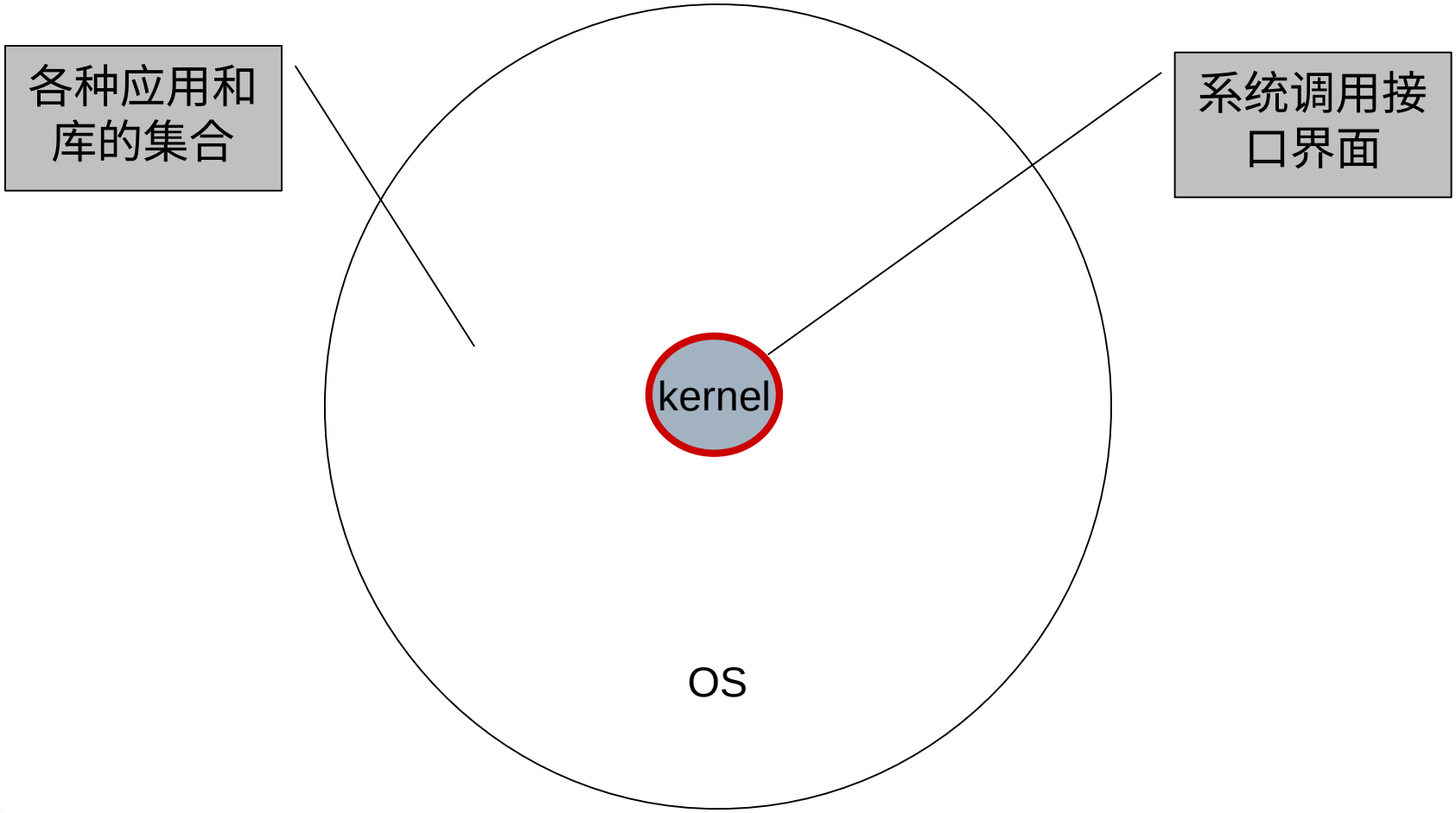
Linux内核

Linux内核体系结构



X86(32, 64), ARM, MIPS, PPC, M86k, Alpha

内核 vs.操作系统

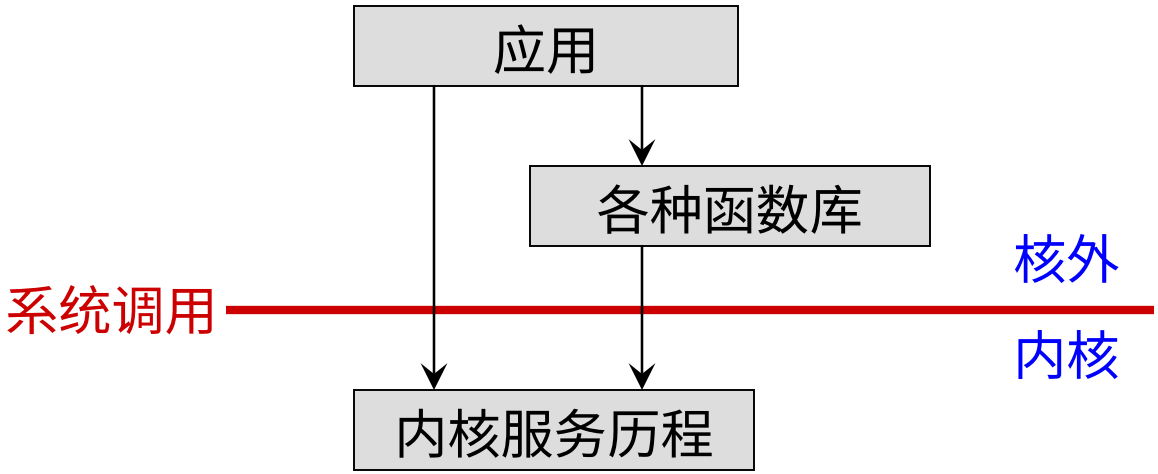


Linux Kernel

- 内核源代码(2.6.19)bz2压缩有40M，展开有200M约16k个文件，编译后可压缩为约1M的核心(vmlinuz)。
- 内核提供了319多个系统调用，这是内核向上层提供服务的接口。
- 内核功能可以在运行过程中以内核模块(kernel module)的方式得以扩充。

从上层(用户层）看Linux内核

- 作为用户与计算机硬件系统之间的接口
- 通过系统调用给用户程序提供各种服务（内核就是服务例程的集合）
- 用户程序不用直接与系统调用打交道，只需使用封装系统调用的API库而已。



为什么选择linux

- 继承了UNIX的优点，有许多改进，是集体智慧的结晶，能紧跟技术发展潮流，具有极强的生命力；
- 通用操作系统
- 符合POSIX标准，各种UNIX应用可方便地移植到Linux下；
- 可自由获得源代码，便于学习，在Linux平台上开发软件成本低

单内核与微内核

单内核与微内核设计之比较

操作系统内核可以分为两大设计阵营：单内核和微内核（第三阵营外内核，主要用在科研系统中，但也逐渐在现实世界中壮大起来）。

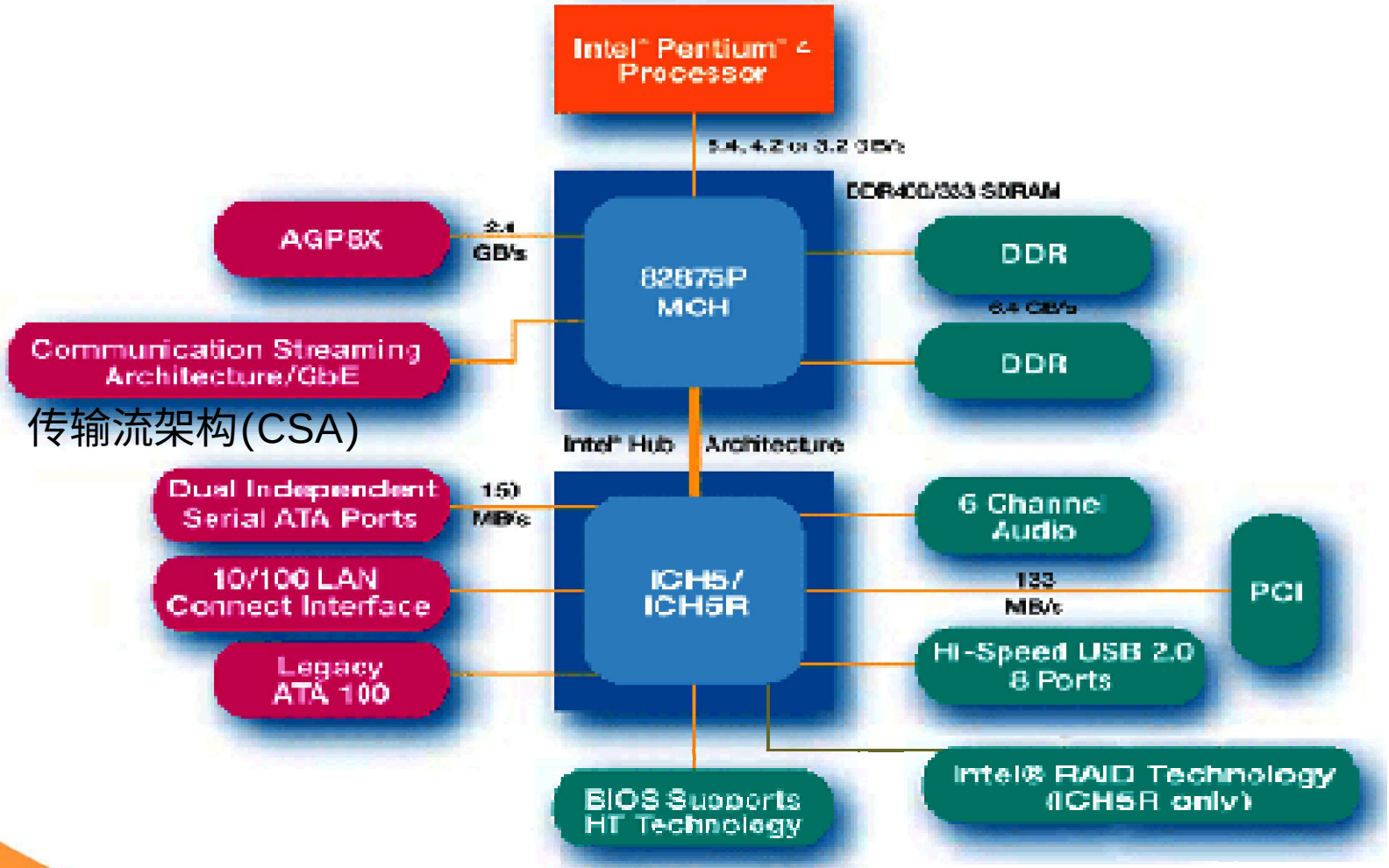
所谓单内核就是把它从整体上作为一个单独的大过程来实现，大多数Unix系统都设计为单模块。微内核的功能被划分为独立 的过程，每个过程叫做一个服务器。

Windows NT内核和Mach（Mac OS X的组成部分）是微内核的典型实例。

Linux是一个单内核，也就是说，Linux内核运行在单独的内核地址空间。不过，Linux汲取了微内核的精华：其引以为豪的是模块化设计、抢占式内核、支持内核线程以及动态装载内核模块的能力。

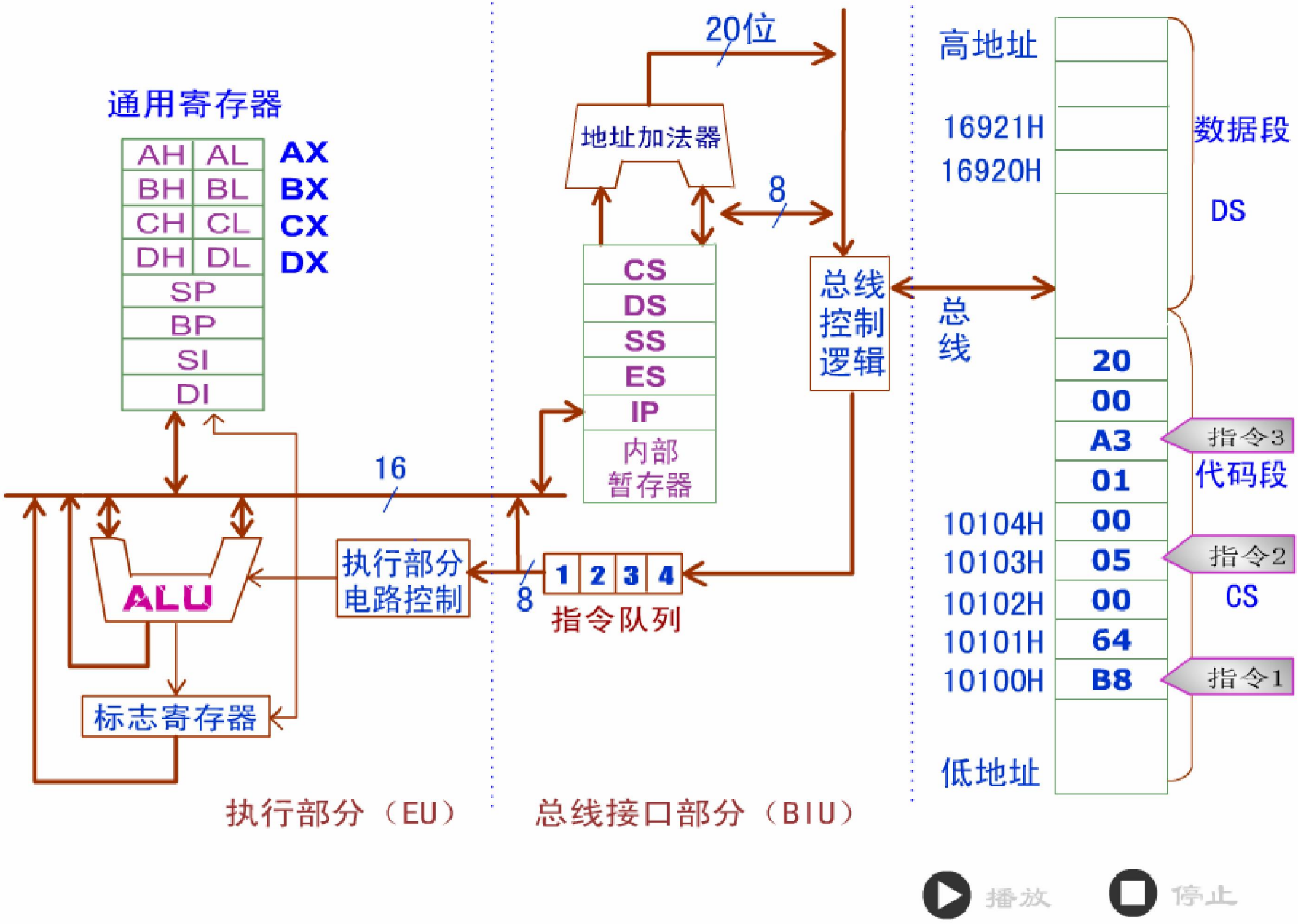
硬件体系结构（x86）

X86 pc 体系结构

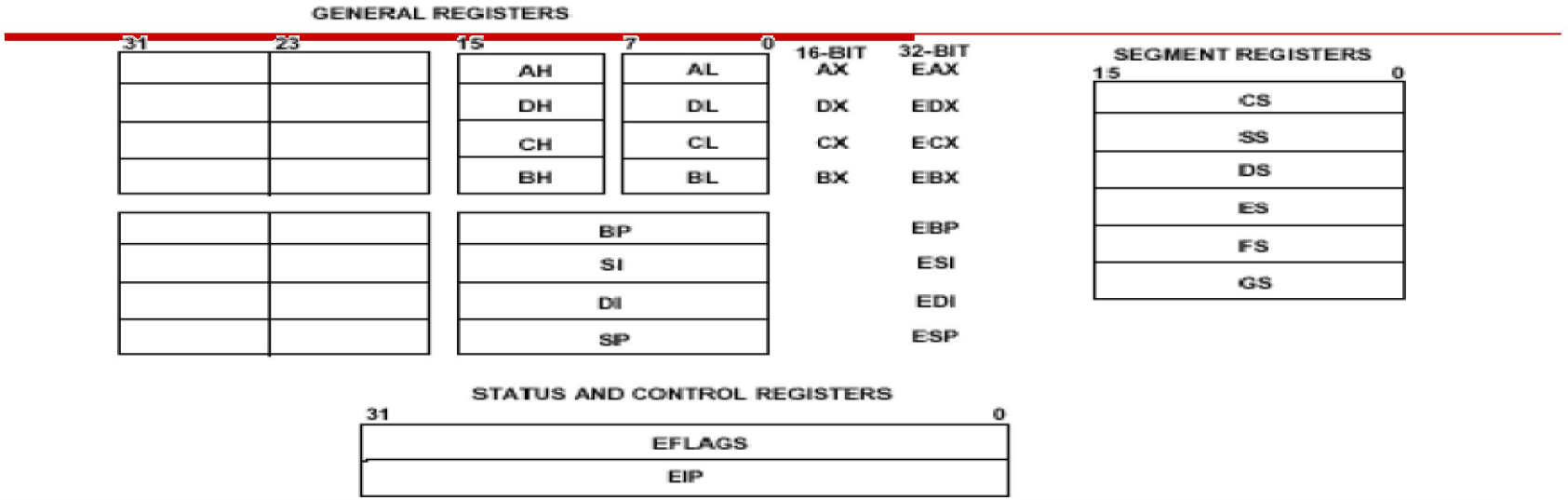


传输流架构(CSA)

IDE(ATA, Advanced Technology Attachment)


通用寄存器
AH AL AX
BH BL BX
CH CL CX
DH DL DX
SP
BP
SI
DI
20位
地址加法器
8
CS
DS
SS
ES
IP
内部
暂存器
总线
控制
逻辑
总
线
16
ALU
执行部分
电路控制
8
1 2 3 4
指令队列
标志寄存器
执行部分（EU）
总线接口部分（BIU）
高地址
16921H
16920H
数据段
DS
20
00
A3
指令3
01
代码段
10104H
00
10103H
05
指令2
CS
10102H
00
10101H
64
10100H
B8
指令1
低地址
播放
停止

i386主要寄存器



调试寄存器，浮点寄存器

80386 内部有 3 个 32 位的控制寄存器 CR0,CR2,CR3，用来控制分页和数学协处理器的操作，保存机器的各个全局状态。其中的 CR0 最重要。

其他处理器

- X86系列
- Motolora系列
- ARM系列
- MIPS系列
- sparc等系列
- Linux源码/arch/...有20多个目录

课程内容

第一部分：内核原理

1. 任务管理
2. 内存管理
3. 文件系统
4. 模块管理
5. 中断管理
6. 时钟管理
7. 网络
8. 进程间通信IPC
9. 其他（如各种设备驱动）

第二部分：驱动开发

1. 如何开发调试内核程序
2. 字符设备
3. 中断服务例程
4. 块设备
5. PCI设备
6. USB设备

后续章节[Linux启动、内存地址空间、系统调用、Linux文件系统...] 请填写您的邮件地址，订阅我们的精彩内容

嵌入式技术交流QQ群：190018652