操作系统

# 体系结构

## 概述





## 内存架构

## 地址

访问内存空间的索引

* 寻址空间

2^位数，如32位，寻址空间为2^32 = 4G

* 物理内存地址空间

处理器提交到总线上用于访问计算机系统中的内存和外设的最终地址，一个计算机系统只有一个物理地址空间

* 线性地址空间

在操作系统的虚存管理之下内个运行的应用程序能访问的地址空间，每个运行的应用程序都认为自己能够独享整个计算机系统的地址空间，如此可让多个运行的应用程序相互隔离

* 逻辑地址空间

是应用程序直接使用的地址空间

## 寄存器

* 通用寄存器

EAX：累加器

EBX: 基址寄存器

ECX:计数器

EDX:数据寄存器

ESI:源地址指针寄存器

EDI:目的地址指针寄存器

EBP:基址指针寄存器

ESP:堆栈指针寄存器

* 段寄存器

CS:代码段(code segment)

DS:数据段(data segment )

ES:附件数据段(extra segment )

SS:堆栈段(stack segment )

FS:附加段

GS:附加段

* 指令指针寄存器EIP：寻址相关
* 标志寄存器：Eflags
* 控制寄存器
* 系统地址寄存器。调试寄存器，测试寄存器

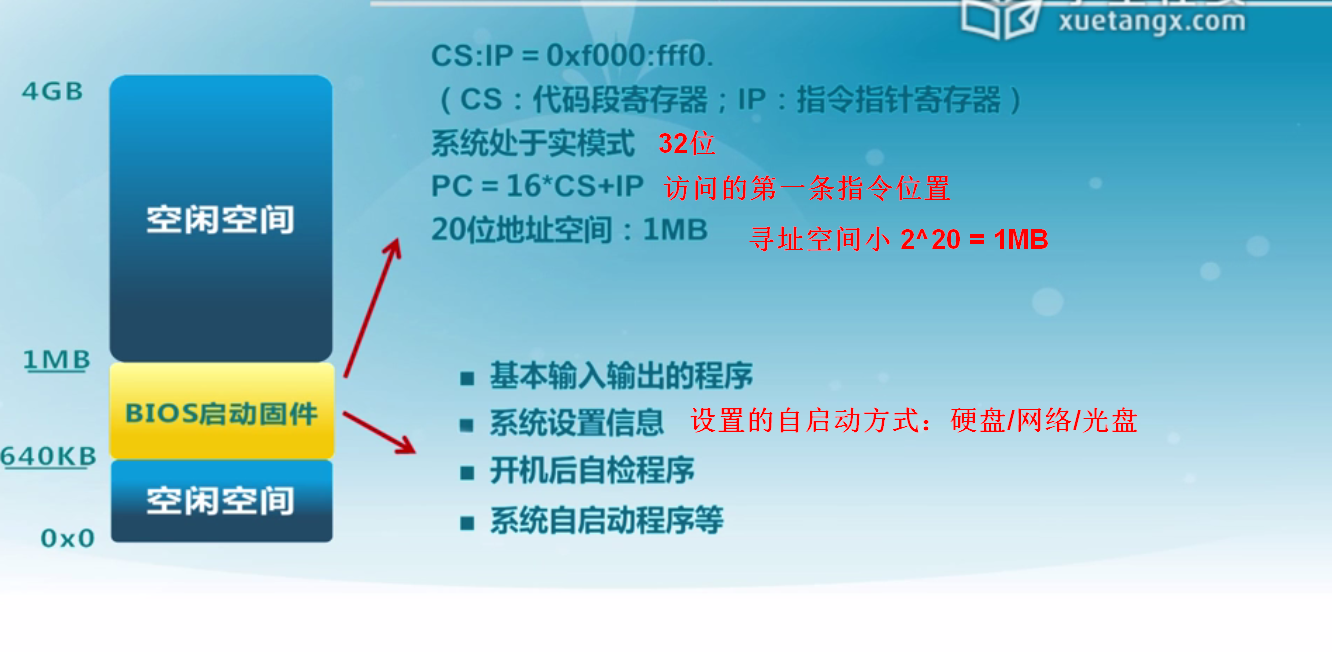
# 启动

## 启动概略

* 内存

RAM(随机访问储存)：通电后系统储存数据，断电后消失：对应下图空闲空间

ROM(随机只读存储)：通电后仍保留一部分启动程序数据：对应下图非空闲空间



* BIOS

导入加载程序，跳转到第一条加载指令

不会直接从BIOS中读取操作系统代码，因为文件系统不一样，不限定死，保持灵活性

I/O功能：字符显示，磁盘扇区读写，检测内存大小，键盘输入

* 加载程序

将操作系统代码数据加载到内存（可以识别不同的文件系统）

跳转到操作系统的其实地址



* 操作系统获得控制权

## 启动详解

### BIOS

* 系统加电-BIOS初始化硬件
* CPU加电后读取第一条指令——跳转指令，跳转到BIOS
* 硬件自检POST,比如内存
* 检测系统中内存和显卡等关键部件的存在和工作状态
* 查找并执行显卡等接口卡BIOS,并执行设备初始化
* 执行系统BIOS，进行系统检测:检测和配置系统中安装的即插即用设备（先关键部件再系统）
* 更新CMOS中的扩展系统配置数据ESCD（系统设备信息表）
* 按指定启动顺序从软盘、硬盘或光驱启动

### 主引导记录

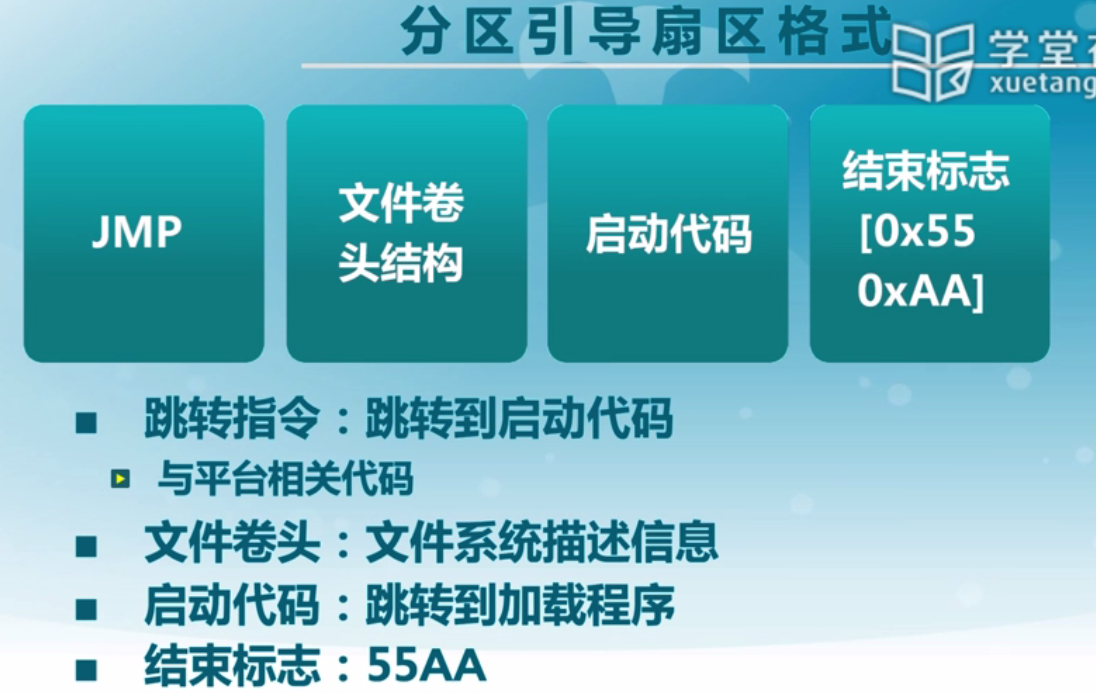
BIOS读取主引导扇区代码

启动代码：检查分区表正确性，加载并跳转到磁盘上的引导程序



### 活动分区

主引导扇区代码读取活动分区的引导扇区代码



### 加载程序

引导扇区代码读取文件系统的加载程序



# 中断、异常和系统调用



## 中断(hardware interrupt )

区分：来自硬件设备的处理请求

### 处理机制

* 硬件

在CPU初始化时设置中断使能标志（CPU初始化完成后才能响应中断）

依据内部或外部事件设置中断标志（记录下有中断事件）

依据中断向量调用相应中断服务例程（定位产生中断的硬件设备）

* 软件

现场保存（编译器）

中断服务处理（服务例程）

清除中断标记（服务例程）

现场恢复（编译器）

* 嵌套

中断和异常是可以出现嵌套的，比如本来有一个中断，但有有另一个中断请求，此时可以按照优先级处理中断，比如电源问题中断和IO中断，优先处理完电源中断。

## 系统调用(system call )

区分：应用系统主动向操作系统发出的服务请求

每个系统调用对应一个系统调用编号

应用程序调用函数库，函数库使用系统函数调用接口，系统调用接口传递系统调用编号，通过软中断进入系统内核（中断向量表），系统调用表根据系统调用编号调用系统实现，返回结果

* 系统调用和函数调用的区别

主要区别：有无堆栈切换和特权级切换

* 系统调用：系统调用主要是在内核态中执行，是可信的应用程序，故而应该有更高的权限，故而会从用户态切换到内核态，会切换堆栈，并且特权级切换到高权限，可以调用设备
* 更安全

更大的开销——用户态到内核态的切换

切换的引导机制

第一次调用时建立内核堆栈

验证参数有效性合法性

内核态访问用户信息需要将内核态映射到用户态的地址空间-更新页面映射权限

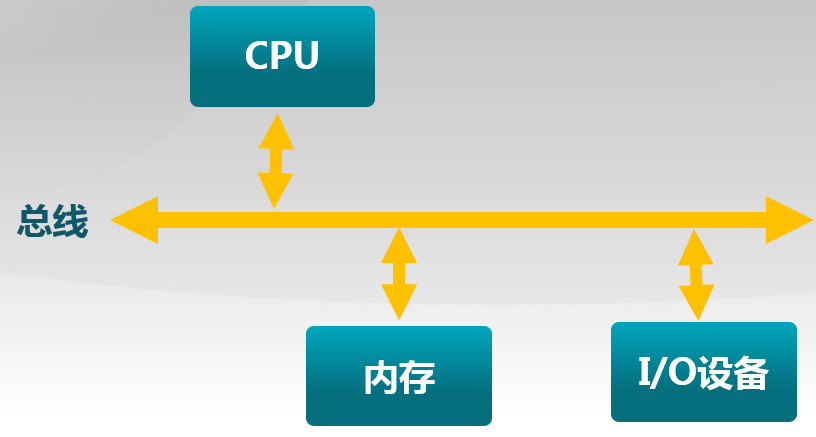
内核态独立地址空间-TLB失效

* 函数调用：常规时没有堆栈切换，不需要切换到可信状态

## 异常(exception )

区分：非法指令或其他原因导致当前指令执行失败（如内存出错）后的处理请求

# 计算机体系结构和内存层次



计算机访问最小访问单位1字节/8bit

32位总线计算机：一次性可读取32位长度数据（4字节），由于地址对齐，可能不能从任意地址开始读取数据，故而32位可能需要两次读取

## CPU



**算术逻辑单元(ALU)**

**寄存器**

**控制逻辑**

**高速缓存**

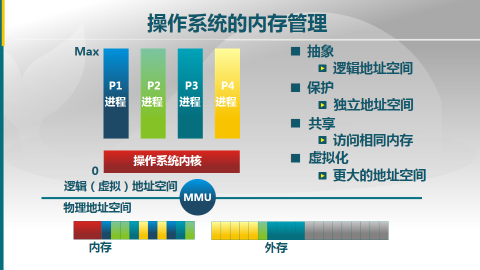
**存储管理单元（MMU）**



CPU中有高速缓存，加快读取速度，由硬件控制，而内存放访问则可由软件控制

CPU 中存在两极缓存:L1缓存L2缓存，若高速缓存未命中，则会从内存中读取，仍未命中，则到外存（虚拟内存/磁盘）中读取

## MMU(存储管理单元)



磁盘访问有扇区编号，每个扇区以512字节为最小单位

为让进程可以共用内存地址（操作系统内核地址），且不互相影响，内存/磁盘和进程间另设了一层存储管理单元，将进程的逻辑地址空间转换成内存/磁盘的物理地址空间

即实际操作代码位于内存，而进程在执行情况不同时，其代码执行代码地址可存于内存和外存，中间转换过程由存储管理单元完成

每个进程的逻辑地址空间可以一样，但物理地址空间经过存储管理单元映射到内存中就不一样

## 操作系统中采用的内存管理方式

* 重定位(relocation)

段地址通过MMU加上偏移地址

* 分段(segmentation)

让进程的地址空间可以不连续（数据、代码、堆栈相对独立）

* 分页(paging)

把内存分成最基本的单位（页中地址连续），若设为一个字节为基本单位，则管理开销太大

* 虚拟存储(virtual memory)

可以让逻辑地址空间大于实际物理地址空间，目前多数系统(如 Linux)采用按需页式虚拟存储

## 地址生成

### 地址空间

* 物理地址空间

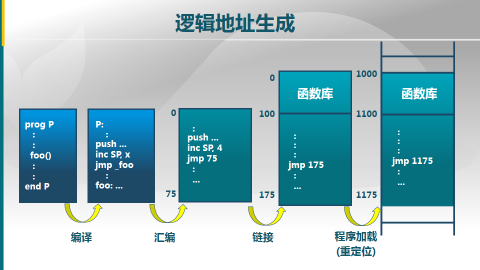
硬件支持的地址空间，地址总线上的地址即为物理地址

32位地址总线，则有32条物理地址总线

* 逻辑地址

CPU运行时，进程使用的地址空间，对应可执行文件中的一段区域，最终会转换成物理地址

### 逻辑地址生成



从局部逐渐到全体的过程

第一次编译：汇编语言，函数引用仍是符号引用，非机器所能识别

第二次编译：函数引用转换成本代码片段的地址引用，编译成二进制机器码

链接：代码引用其他模块和函数库，所有重排成线性序列，地址改变

程序加载（重定位）：运行时地址，地址再次改变，成为逻辑地址

### 地址生成时机和限制

* 编译时：如不能安装软件的功能手机

假设起始地址已知

如果起始地址改变，必须重新编译

* 加载时：如智能手机

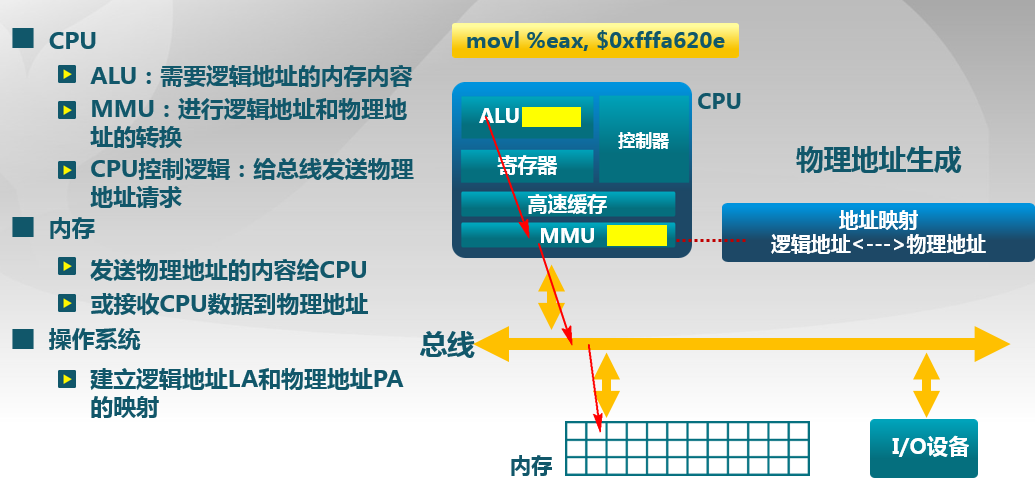
如编译时起始位置未知，编译器需生成可重定位的代码(relocatable code)

加载时，生成绝对地址

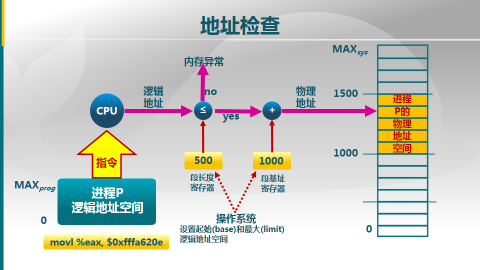
* 执行时，灵活

执行时代码可移动

需地址转换(映射)硬件支持



ALU：算数逻辑单元



## 连续内存分配和内存碎片

### 连续内存分配

* 连续内存分配

给进程分配一块不小于指定大小的连续的物理内存区域

* 内存碎片

空闲内存不能被利用

* 外部碎片-进程间

分配单元之间的未被使用内存

* 内部碎片-进程内

分配单元内部的未被使用内存

取决于分配单元大小是否要取整



### 动态分区分配

当程序被加载执行时，分配一个进程指定大小可变的分区(块、内存块)，分区的地址是连续的

* 动态分区分配策略
  + 最先匹配(First-fit)

空闲分区列表按地址顺序排序

分配过程时，搜索一个合适的分区

释放分区时，检查是否可与临近的空闲分区合并

**·优点**

在高地址空间有大块的空闲分区

**·缺点**

外部碎片较多

分配大块时网后找，程序运行越久就较慢

* + 最佳匹配(Best-fit)

空闲分区列表按照大小排序

分配时，查找一个合适的分区

释放时，查找并且合并临近的空闲分区（如果找到）

**·优点**

大部分分配的尺寸较小时，效果很好

可避免大的空闲分区被拆分

可减小外部碎片的大小

**·缺点**

释放分区较慢（临近合并）

容易产生很多无用的小碎片

* + 最差匹配(Worst-fit)

空闲分区列表按由大到小排序

分配时，选最大的分区

释放时，检查是否可与临近的空闲分区合并，进行可能的合并，并调整空闲分区列表顺序（慢）

**·优点**

中等大小的分配较多时，效果最好

避免出现太多的小碎片

**·缺点**

释放分区较慢（合并）

容易破坏大的空闲分区，因此后续难以分配大的分区

## 碎片整理