计算机系统漫游

> 程序的执行需要哪些软硬件支持?

什么是计算机系统?

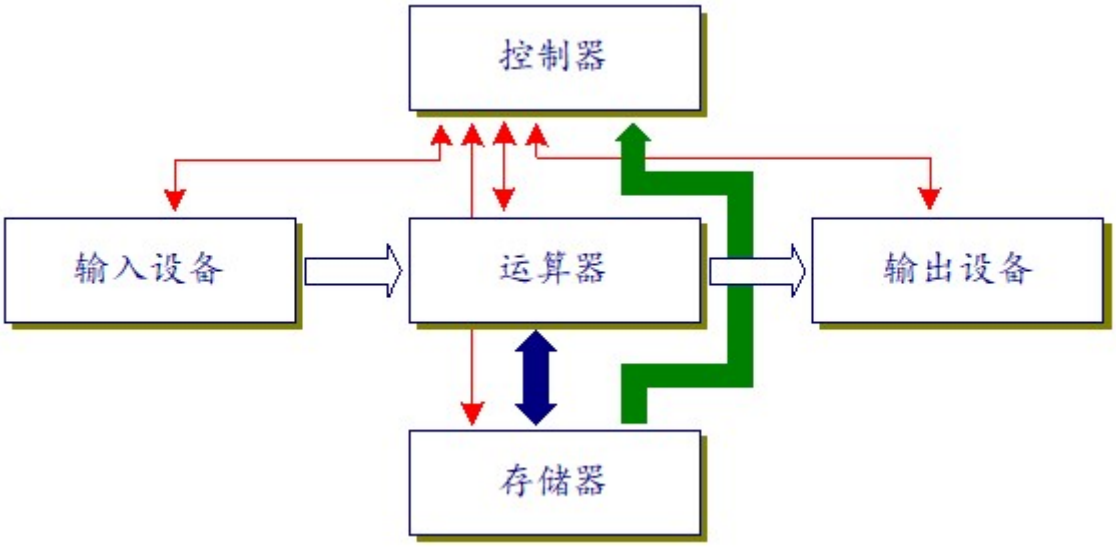
- 软件
  - 应用；算法；编程；操作系统
- 指令集架构
- 硬件
  - 微架构；功能部件；电路；器件

虚拟机观点下的计算机系统

- 第五级-虚拟机器：应用语言及其M5，具有L5机器语言（应用语言）
- 第四级-虚拟机器：高级语言
- 第三级-虚拟机器：汇编语言
- 第二级-虚拟机器：操作系统
- 第一级-实际机器：传统机器
- 第零级-实际机器：微程序机器

计算机的组织结构

冯诺依曼结构

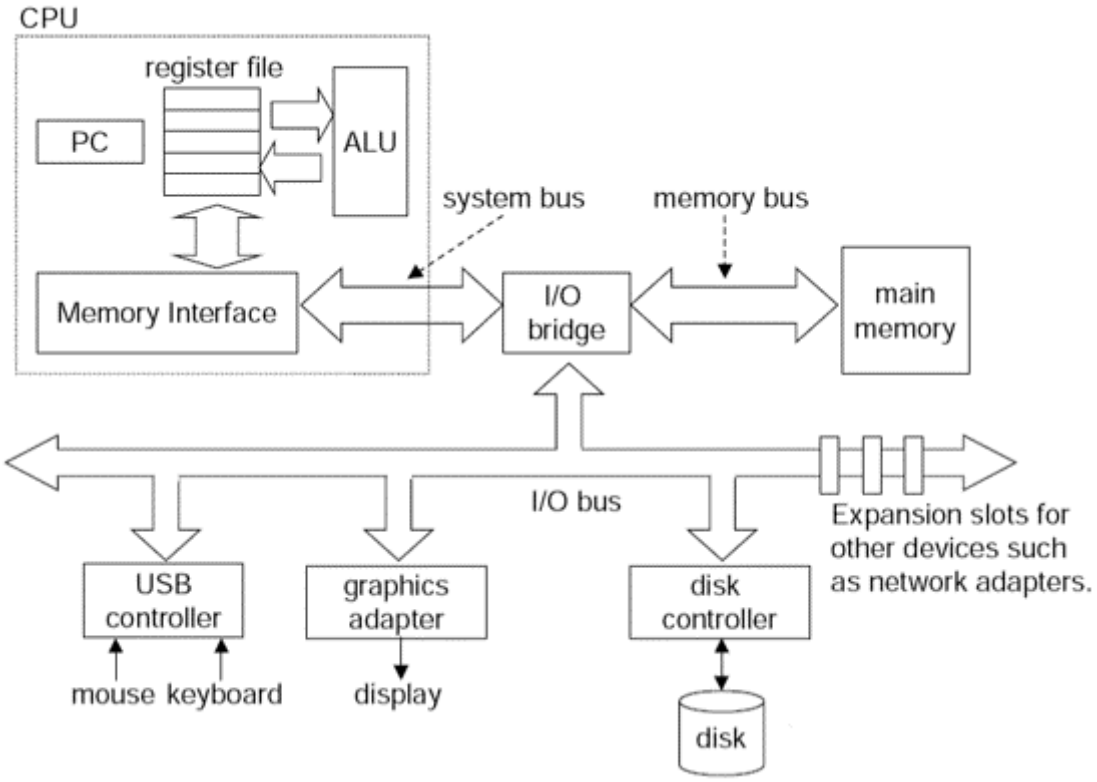


特点:

- 存储程序思想：计算机系统由硬件、软件构成

- 存储程序思想：计算机系统由硬件、软件组成
- 二进制：数据和程序指令以二进制编码，存储在内存中
- 顺序执行：程序中的指令依次被执行。

哈佛结构



特点：以主存为中心(DMA技术)、标准化总线、层次化存储体系

程序的翻译

程序从源程序转化成为目标程序是通过编译系统构成的

1. 源程序通过预处理器进行预处理，输出修改后的源程序
2. 进入编译器，输出汇编程序
3. 进行汇编器，输出二进制文件(可重定位目标程序)
4. 和外部库(例如使用了printf函数就要与print.o文件结合)连接，输出可执行目标程序(二进制)

程序的执行

机械特性

存储设备的机械特性：存储容量越大，访问速度越慢，处理器访问的开销就越大

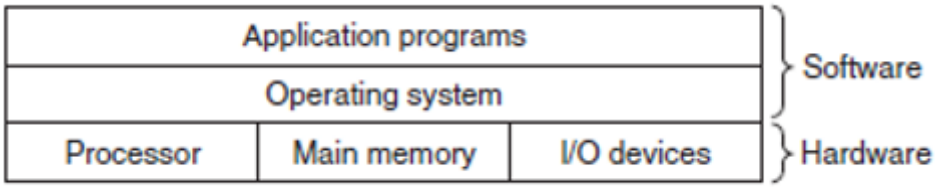
内存：容量小（G） ,速度快(10GB/s)

磁盘：容量大（T），速度慢(100MB/s)

高速缓存-见第六章

操作系统

> 操作系统是应用软件和计算机硬件之间的桥梁
>
> 优点
>
> - 防止应用程序滥用操作系统
> - 向应用程序提供统一的机制完成对底层硬件的访问



- 进程：是对处理器、内存、外设的抽象
  - 虚拟内存：是对存储器和外设的抽象
    - 文件：是对I/O的抽象

进程：对正在运行程序的抽象
用户角度：独占系统
系统角度：多个任务并发执行
线程：进程的一个执行单元，一个进程中可以包含多个线程

Amdahl定律

> 系统某个部分被加速时，其对系统整体性能的影响主要取决于该部分的重要程度和加速速度

公式：$S = \frac{T_{old}}{T_{new}} = \frac{1}{(1-\alpha)+\frac{\alpha}{k}}$

> 假设一台计算机的所有计算任务中 I/O 处理占 20%，当计算机中 CPU 性能提高为原来的 10 倍，而I/O 性能提高为原来的 2 倍时，系统总体性能会出现什么变化?

> 计算机执行某测试程序，其中含有大量浮点数据的处理操作，为提高性能可以采用两种方案，一是采用硬件实现求浮点数平方根(FPSQR)的操作，可以使该操作的速度提高10倍；另一种方案是提高所有浮点数据操作(FP)的速度，使其加快2倍。同时已知FPSQR操作时间占整个测试程序执行时间的20%，而FP操作占整个执行时间的50%，现比较两种方案

并发和并行

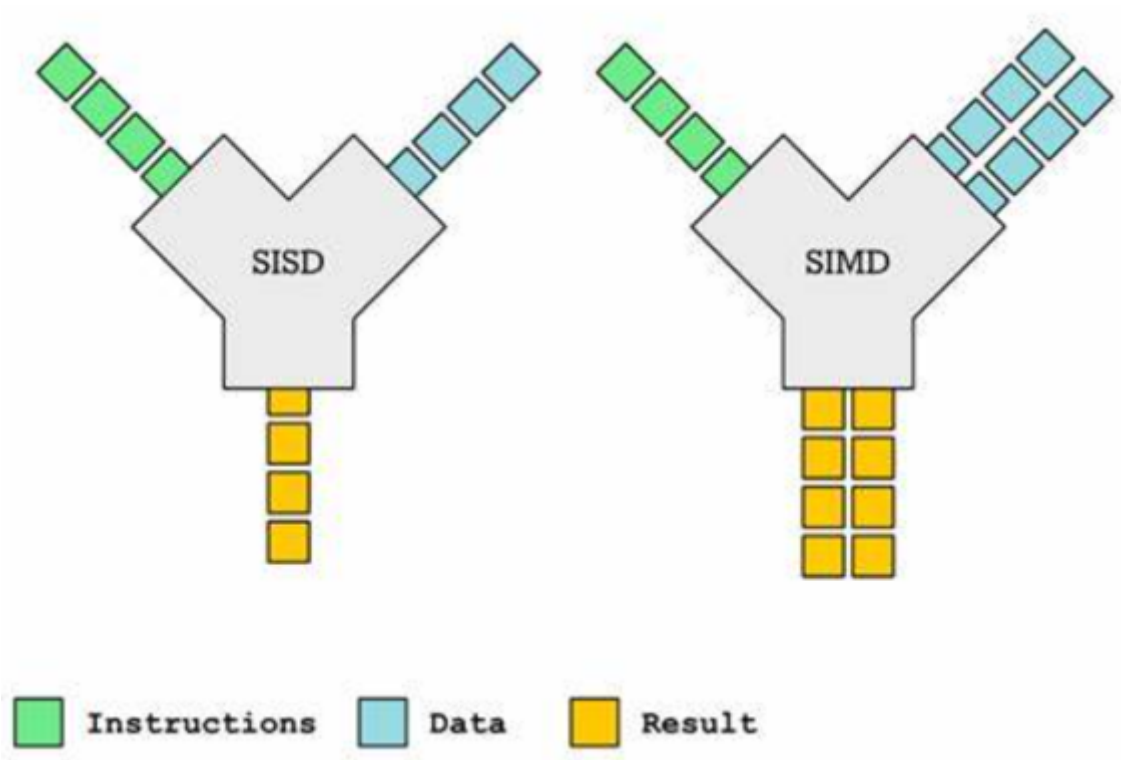
并发：指多个任务在同一个时间段内快速地轮换执行

- 线程级并发
  - 单处理器系统：多个任务轮流与系统交互
  - 多处理器系统：每个处理器完成一个任务。多个任务可以并行
    - 多核处理器：通过增加 CPU 的核心数，可以提高系统的性能
    - 超线程：同时多线程，允许cpu同时执行多个控制流

并行：是指多个任务同时进行，是真正意义上的同时执行

- 指令级并行：处理器可以同时执行多条指令的属性

单指令单数据和单指令多数据并行



习题

对齐：不同数据存放有要求

字节大小-任意

字数据-最低有效位等于0

32位数据-起始地址必须是四的倍数，最低两位是00

64位-最低三位是000

栈破坏检测

栈随机化

TLB加快页表访问速度

信息表示和处理

指令中信息

指令格式：操作码+操作数

操作数类型：

- 数值型
- 其他型：堆栈、字符串、向量等

类型确定方法：指令操作码确定

信息存储

- 以字节为单位，每个字节具有唯一地址
- 所有有可能的地址集合称为虚拟地址空间
- 每个字节由8个二进制位构成：1B=8bit

字节序和对齐

大端小端字节序：字节存储，字访问。存储数据大于字节，需要存放顺序

对齐：数据存放地址的要求，按照数据类型大小的整数倍来进行存储

- 保证存储访问的效率，否则访问双字数据有可能出现多次内存访问

位运算和逻辑运算

位运算：

- 非：就是每一位与进制取反
  - 二进制：~1=0
  - 十六进制：~4=B
- 和：都为1则1，否则为0
  - 二进制：11&10=10
  - 十六进制：69&55=41
- 或运算：对应位置只要有1就为1

逻辑运算：

-

整数表示

- 表示：源码，反码，补码，移码
- 四则运算
  - 乘法---移位运算和加法运算代替
  - 除法---数据最后的舍入，加入偏置值

浮点数表示

- 基本格式：符号，阶码，尾数
  - 移码表示阶码：8位阶码---指数+127
  - 尾数用原码表示(单精度、双精度)

> bias=$2^{n-1}-1$

规格化

E=e-bias

M=1+f

非规格化

阶码全为0： E=1-bias，M=f

阶码全为1： f全为0则无穷，否则Nan

舍入

向上：向大于x的值舍入

向下：向小于x的值舍入

向零：向接近零的摄舍入

向偶：向最接近的值舍入，1.5则向偶数舍入

处理器体系结构

寄存器功能

寄存器地址	功能
%rax	存放返回值
%rbx	被调用者保存地址
%rcx	存放第四个参数
%rdx	存放第三个参数
%rsi	存放第二个参数
%rdi	存放第一个参数
%rbp	被调用者保存
%rsp	存放栈指针
%r8--%r9	第5-6个参数
%r10-%r11	调用者保存
%r12--%r15	被调用者保存

> 调用者保存：
> 函数A在调用函数B之前，提前保存寄存器rbx的内容，执行完函数B之后，再恢复寄存器rbx原来存储的内容
>
> 被调用者保存：
> 函数B在使用寄存器rbx之前，先保存寄存器rbx的值，在函数B返回之前，先恢复寄存器rbx原来存储的内容

操作数与寻址

操作数类型：

- 立即数：用$表示，传送指令：movq $0x4,%rax
- 寄存器：直接调用寄存器，传送指令：movq %rax,%rdx
- 存储器：使用括号将寄存器括起来表示值，传送指令：movq (%rax),%rdx

各类指令功能

> b：1； w：2； l：4； q：8；

传送类指令

把数据从源位置复制到目的位置mov包括movb,movw,movl,movq

出栈入栈指令：堆栈也可以完成数据传送的工作，它是遵循先进后出规则的一个内存区域。它的地址向下增长，栈顶元素的地址最小

运算类指令

描述	指令	效果
加载有效地址	leaq S,D	D<--&S
加1	INC D	D<--D+1
减1	DEC D	D<--D-1
取负	NEG D	D<---D
取反	NOT D	D<--~D
加法	ADD S,D	D<--D+S
减法	SUB S,D	D<--D-s
乘法	IMUL S,D	D<--D*S
异或	XOR S,D	D<--D^S
或	OR S,D	D<--D|S
与	AND S,D	D<--D&S
算数左移	SAL k,D	D<--D<<k
逻辑左移	SHL k,D	D<--D<<k
算术右移	SAR k,D	D<--D>>$_A$k
逻辑右移	SHR k,D	D<--D>>$_L$k

加载有效地址：源操作数是地址表达式，但不直接引用内存，不敌操作数存放表达式形成的地址

一元操作：只有一个操作数，因此该数即是源操作数又是目的操作数，可以说寄存器也可以是内存地址

二元操作数：
第一个操作数是源操作数，可以是立即数、寄存器或者内存地址；
第二个操作数既是源操作数也是目的操作数， 可以是寄存器或者内存地址，但不能是立即数

移位操作：
左移指令SAL和SHL，二者的效果是一样的，都是在右边填零
右移指令包括算术右移和逻辑右移
算术右移算术右移需要填符号位

逻辑右移，逻辑右移需要填零

> 对于移位量k，可以是一个立即数，或者是放在寄存器cl中的数，对于移位指令只允许以特定的寄存器cl作为操作数，其他寄存器不行

条件码

条件码	作用
CF	进位标志,可以用来检查无符号数操作的溢出，产生进位置1
ZF	零标志，当最近操作的结果等于零时,置1
SF	符号标志，当最近的操作结果小于零时，置1
OF	溢出标志，针对有符号数，最近的操作导致正溢出或者负溢出时，被置1

> Leaq对条件码不起作用

指令	同义名	效果	设置条件
sete D	setz	D ← ZF	相等/零
setne D	setnz	D ← ~ZF	不等/非零
sets D		D ← SF	负数
setns D		D ← ~SF	非负数
setg D	setnle	D ← ~(SF ^ OF) & ~ZF	大于（有符号>）
setge D	setnl	D ← ~(SF ^ OF)	大于等于（有符号>=）
setl D	setnge	D ← SF ^ OF	小于（有符号<）
setle D	setng	D ← (SF ^ OF) | ZF	小于等于（有符号<=）
seta D	setnbe	D ← ~CF & ~ZF	超过（无符号>）
setae D	setnb	D ← ~CF	超过或相等（无符号>=）
setb D	setnae	D ← CF	低于（无符号<）
setbe D	setna	D ← CF | ZF	低于或相等（无符号<=）

比较指令

描述	指令	基于
比较	cmp S_1, S_2	$S_2 - S_1$
测试	TEST S_1, S_2	$S_1 \& S_2$

跳转指令

将当前指令的执行从当前位置切换到新位置，可以直接进行或根据状态码状态

指令	同义名	跳转条件	描述
jmp *Label*		1	直接跳转
jmp **Operand*		1	间接跳转
je *Label*	jz	ZF	相等/零
jne *Label*	jnz	~ZF	不相等/非零
js *Label*		SF	负数
jns *Label*		~SF	非负数
jg *Label*	jnle	~(SF ^ OF) & ~ZF	大于（有符号>）
jge *Label*	jnl	~(SF ^ OF)	大于或等于（有符号>=）
jl *Label*	jnge	SF ^ OF	小于（有符号<）
jle *Label*	jng	(SF ^ OF) | ZF	小于或等于（有符号<=）
ja *Label*	jnbe	~CF & ~ZF	超过（无符号>）
jae *Label*	jnb	~CF	超过或相等（无符号>=）
jb *Label*	jnae	CF	低于（无符号<）
jbe *Label*	jna	CF | ZF	低于或相等（无符号<=）

条件分支的实现

控制的条件转移：比较指令+跳转指令

数据的条件转移：

指令	同义名	传送条件	描述
cmove *S, R*	cmovz	ZF	相等/零
cmovne *S, R*	cmovnz	~ZF	不相等/非零
cmovs *S, R*		SF	负数
cmovns *S, R*		~SF	非负数
cmovg *S, R*	cmovnle	~(SF ^ OF) & ~ZF	大于（有符号>）
cmovge *S, R*	cmovnl	~(SF ^ OF)	大于或等于（有符号>=）
cmovl *S, R*	cmovnge	SF ^ OF	小于（有符号<）
cmovle *S, R*	cmovng	(SF ^ OF) | ZF	小于或等于（有符号<=）
cmova *S, R*	cmovnbe	~CF & ~ZF	超过（无符号>）
cmovae *S, R*	cmovnb	~CF	超过或相等（无符号>=）
cmovb *S, R*	cmovnae	CF	低于（无符号<）
cmovbe *S, R*	cmovna	CF | ZF	低于或相等（无符号<=）

> 表达式的计算复杂度高时，不适合用条件传送

循环

do-while

> 先执行do，在进行条件判断，满足则跳回goto循环起始位置

while

> 先跳，跳到条件判断，满足则再跳到do位置

for

> 将for的参数初始化在循环外赋值，限制条件放在while里，改写成while再进行

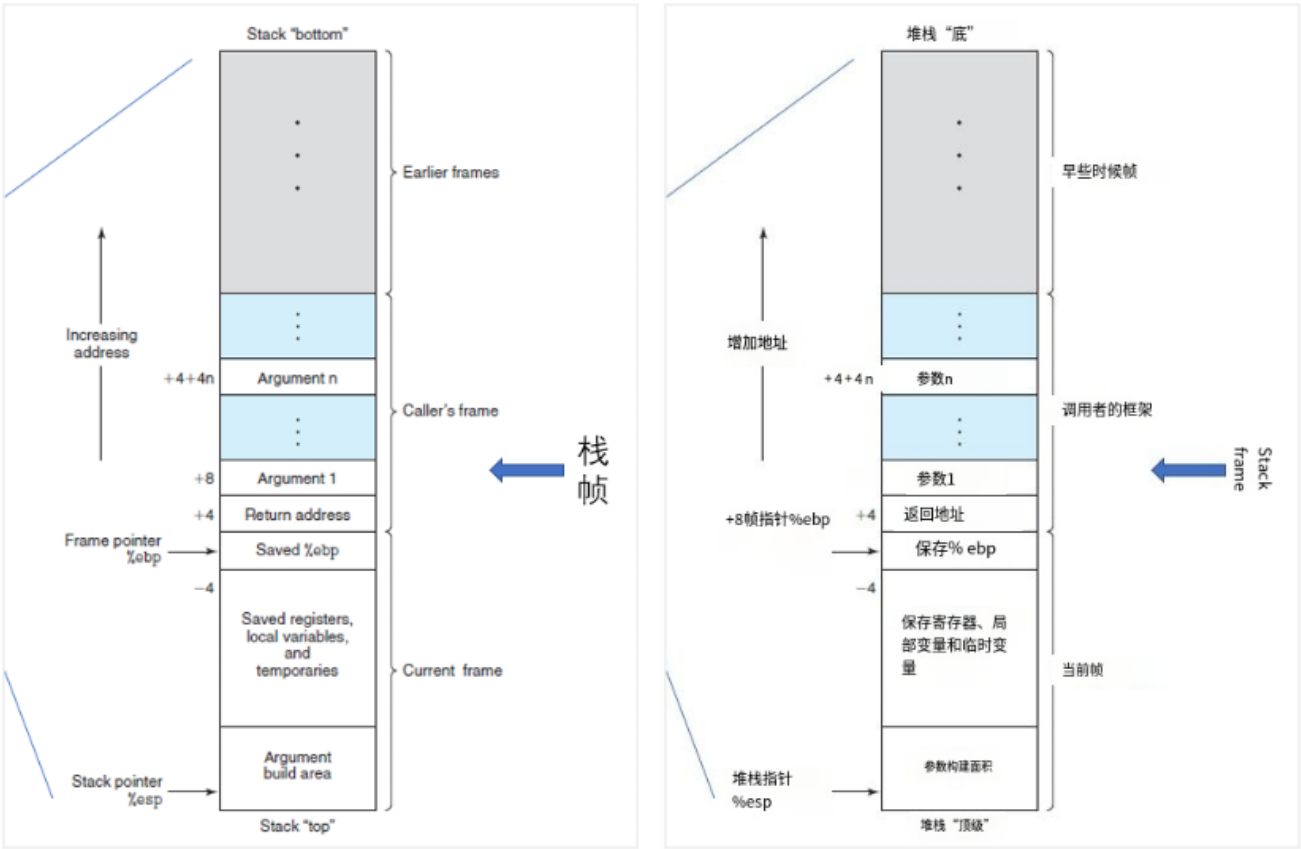
switch

> 先进性判断，然后跳转goto到对应的case中，否则跳转到default处，break即跳转到最后

跳转表

过程

栈帧：



转移控制：函数P调用函数Q，把返回地址压入栈中，指明当函数Q执行结束返回时要从函数P的哪个位置继续执行，由call指令执行

Call指令：call label

- 返回地址压栈
- 跳转到label指示的地址

Ret指令：ret

- 弹出返回地址
- 转向返回地址

数据传递

> 如果一个函数的参数数量大于6，超出的部分就要通过栈来传递
>
> movq n*4(%rsp)，%rax

栈上的局部存储

> 当代码中对一个局部变量使用地址运算符‘&’时，需要在栈上为这个局部变量开辟相应的存储空间

寄存器的局部存储空间

当过程P调用Q时，P为调用者，Q为被调用者

调用者保存：调用者将临时值保存在自己的栈帧中
被调用者保存：被调用者在使用前将临时值保存在自己的栈帧中，返回时进行恢复

数组

数组T A[L]：

- 数据类型T和长度L
- 需要给数组分配连续空间大小为L*sizeof(T)字节
- 任何字节的指针都是8字节

嵌套数组T A[R][C]：嵌套数组中行优先

- A是二维数组
- R是行数，C是列数
- T是A中元素的数据类型
- 数据连续空间大小为R*C*T字节
- 地址访问为：A+i*(C*T)+J*T=A+(i*C+j)*K
  - 数组起始地址A，加上行起始地址，再加上列起始地址

定长数组和变长数组

结构体

结构：将不同类型的对象集中到一个对象中

结构体的所有组成部分存储在内存一段连续的区域中存放的顺序按照声明的顺序
指向结构体的指针是结构体第一个字节的地址

对齐

数据以数据类型的整数倍分配空间

结构体数据的对齐：

- 结构体内部：满足每个数据的对齐条件
- 整个结构体：结构体的对齐按照最长元素长度K来决定

缓冲区溢出

> 内存越界导致缓冲区溢出

解决溢出方法：

- 栈随机化：栈的位置在程序每次运行时都有变化
- 栈破坏检测：栈保护机制：在缓冲区与栈保存的状态值之间存储一个特殊值，这个特殊值被称作金丝雀值，这个数值是程序每次运行时随机产生的，在函数返回之前，检测金丝雀值是否被修改来判断是否遭受攻击
- 限制可执行代码区域

存储器

存储器概述

随机访问存储器RAM

静态随机访问存储器SRAM

特点：

- 速度快，接近CPU
- 只要通电，就能保持内部存储的数据
- 对干扰不敏感，不需要刷新
- 同样面积的芯片容量较小
- 价格高
- 一般用于构成cache

动态随机访问存储器DRAM

特点：

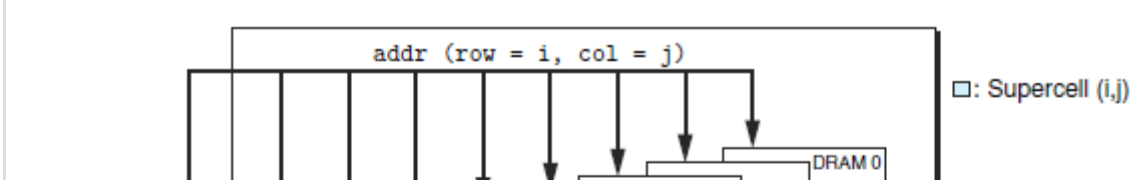
- 速度比SRAM慢，容量比SRAM更大
- 对干扰敏感，会产生漏电，需要刷新
- 异步方式，随时响应控制输入的变化

增强的DRAM：

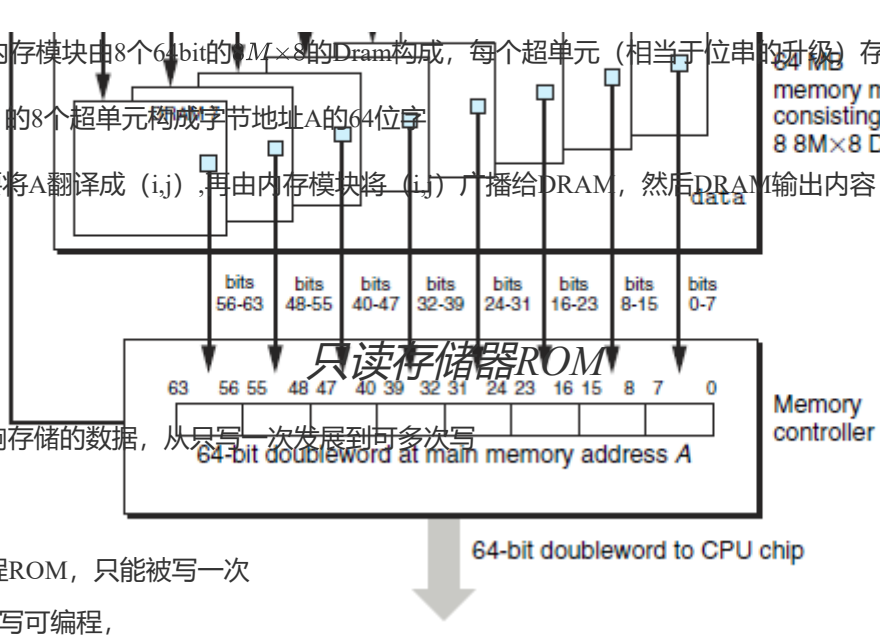
- 同步动态随机存取内存SDRAM：
  - SDRAM有一个同步接口，在响应控制输入前会等待一个时钟信号，这样就能和计算机的系统总线同步，实现更为复杂的操作模式，更高的速率
  - SDRAM在一个时钟周期内只传输一次数据，它是在时钟上升期进行数据传输
- 双倍速率同步动态随机存储器DDR SDRAM：
  - DDR是一个时钟周期内可传输两次数据，也就是在时钟的上升期和下降期各传输一次数据

内存模块

DRAM被封装在内存模块中，多个内存模块连接到内存控制器上，聚合成主存



> 上图中一个内存模块由8个64bit的$M\times8$的Dram构成，每个超单元（相当于位串的升级）存储主存的一个字节
>
> 地址为（i,j）的8个超单元构成字节地址A的64位字
>
> 读取A，需要将A翻译成（i,j），再由内存模块将（i,j）广播给DRAM，然后DRAM输出内容



只读存储器ROM

特点：掉电不影响存储的数据，从只写一次发展到可多次写

代表：

- PROM：可编程ROM，只能被写一次
- EPROM：可擦写可编程，
- EEPROM：电子可擦除。
- Flash memory：基于EEPROM，代表U盘，存储卡，固态硬盘
- NOR Flash存储器（在芯片内执行，因此应用程序可以直接在Flash中运行，而不必读入系统RAM）
- NAND Flash存储器（存储容量更大，擦写速度更快）

磁盘存储器

特点：

- 容量大
- 速度较慢
- 每位价格较低

机械磁盘

扇区访问时间：$T_{access}=T_{seek}+T_{rotation}+T_{transfer}$（寻道时间+旋转时间+传送时间）

固态硬盘

程序的局部性

局部性：程序将要引用的数据或者指令在地址上邻近或者就是正在访问的地方

时间局部性：当前引用的内容很可能在不远的将来再次被访问

空间局部性：即将访问的内容与当前访问的内容在地址上是邻近的

存储设备的机械特性

机械特性：容量越大，速度越慢，位价格越低；速度越快，容量越小，位价格越高

由于程序的局部性，程序更频繁的访问某一层次上的存储设备，因此下层的设备可以更慢速一些，容量可以更大，每位价格更低。因此将特性不同的存储设备结合起来刚好可以满足程序局部性的特征，形成存储体系。

存储器层次结构

定义：两个和两个以上的速度、容量、价格各不相同的存储器用硬件、软件或软硬结合的方法连接起来的系统

核心思想：

- 缓存：层次结构中的每一层都缓存来自较低一层的数据块
- 缓存的工作模式：
  - 将缓存划分成相同数据块, 在相邻层之间传送块, 判断命中与不命中, 并处理它们, 由硬件、软件或两者结合的方式实现
  - 位于每个k层的速度更快、容量更小的设备作为k+1层的缓存

特点：速度接近最快的存储器；容量与容量最大的存储器相等或接近；单位价格接近于最便宜的存储器

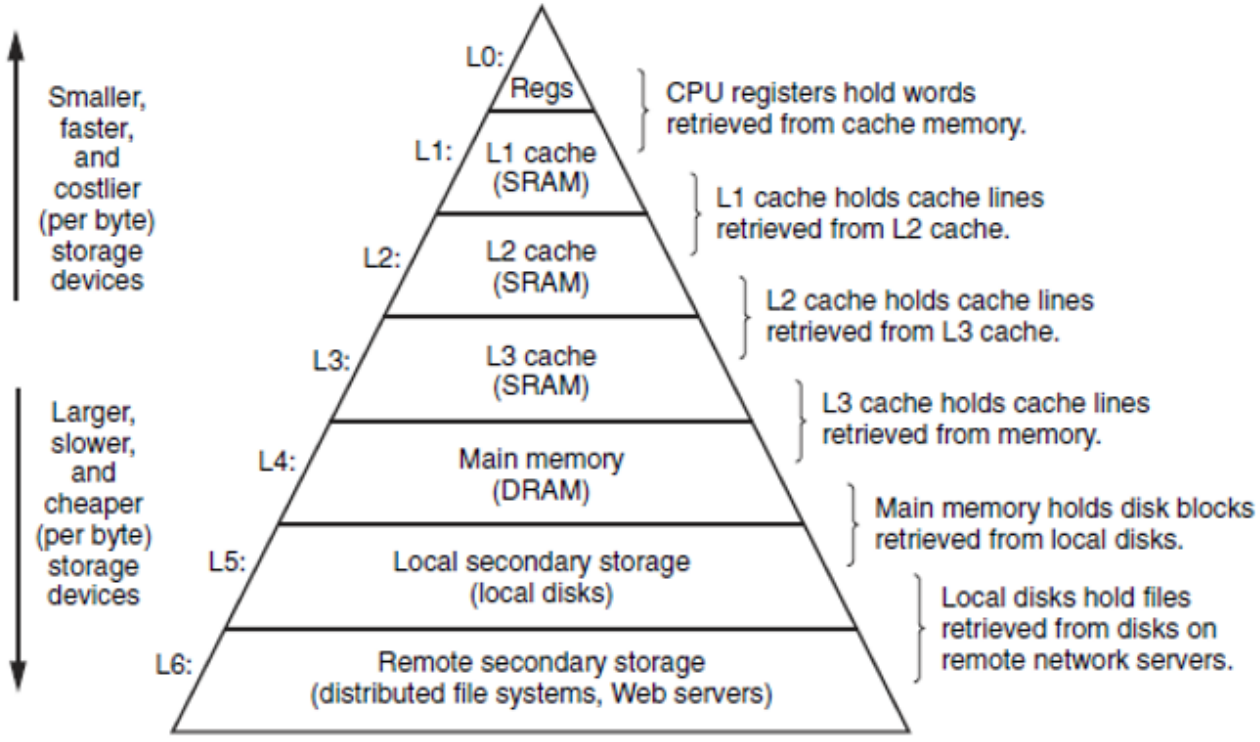


Figure 6.23 **The memory hierarchy.**

主存-辅存

Cache-主存

交叉访问存储器

关键问题

命中：当访问第k+1层的数据对象时，刚好该对象缓存在第k层

不命中：当访问第k+1层的数据对象时，该对象不在第k层缓存中

- 冷不命中(cold miss): 空缓存(冷缓存cold cache)造成不命中
- 冲突不命中(conflict miss): 限制性映像策略造成不命中
- 容量不命中(capacity miss): 缓存容量小于工作集造成不命中

替换：不命中时，需要从k+1层中将目标数据对象缓存到k层，如果k层满了，则替换掉现有缓存块

- 冲突：需要替换的现象
- 牺牲块：被替换掉的数据缓存块

高速缓存存储器Cache

典型结构

使用元组(S,E,B,m)表示：Set组，Cache Line行数，数据块包含字节，数据地址

每个cache被划分为多个Set，每个Set包含一个或多个CacheLine，每个CacheLine由有效位Valid，标记Tag和数据块Block组成

- 有效位：1个bit，表示当前存储的信息是否有效
- 标记
- 数据块：一小部分内存数据的副本，大小是B

cache容量$C = S * E * B$

地址映像分类

直接映像

E=1：每一组Set只有一行Cache Line

1. 从L1.Cache中查询目标数据，命中则返回
2. 命中判断
    1. 组选择：根据组索引值
    2. 行匹配：每个Set只有一个Cache Line，对比Cache Line中的标记和目标地址的标记是否一致
    3. 字选择：从数据块的什么位置抽取数据，根据偏移量判断数据起始地址
3. 不命中：从下一层取出被请求的块，直接映射Set只有一行，就是直接替换即可

抖动：出现冲突不命中，导致Cache Line的替换

- 解决：数据填充

组相联映射

$1 < E \leq C/B$：每个组Set至少有一个Cache Line

1. 从L1.Cache中查询目标数据，命中则返回
2. 命中判断
    1. 组选择：根据组索引值
    2. 行匹配：每个Set有多个Cache Line，遍历Set中每一行寻找有效位为1且标记为相同的cache line
    3. 字选择：从数据块的什么位置抽取数据，根据偏移量判断数据起始地址
3. 不命中：从下一层取出被请求的块，进行替换
    1. 优先替换有效位为0的行
    2. 使用替换策略：随即替换、LFU最不常使用、LRU最近最少

特点：

- 硬件实现很简单, 不需要相联访问存储器
- 访问速度也比较快, 实际上不做地址变换
- 块的冲突率较高

全相联映射

S=1 E=C/B：只有一个组Set

- C表示cache容量，B表示数据块大小

全相联只有一个组，因此不需要组索引位进行组选择

全相联适合容量较小的高速缓存

Cache一致性问题

- CPU写Cache的时候，修改了Cache的内容，但主存对应单元的内容与cache单元内容不一致；
- I/O设备读入数据到主存，修改了主存某单元的内容，而Cache中的内容还没有修改，也会使得cache与主存的单元内容不一致

通常**写分配和写回搭配**，**写不分配和写穿透搭配**

写命中

写直达法write-through：写操作时，利用CPU与主存的通路，将块写入Cache中，也写入到主存

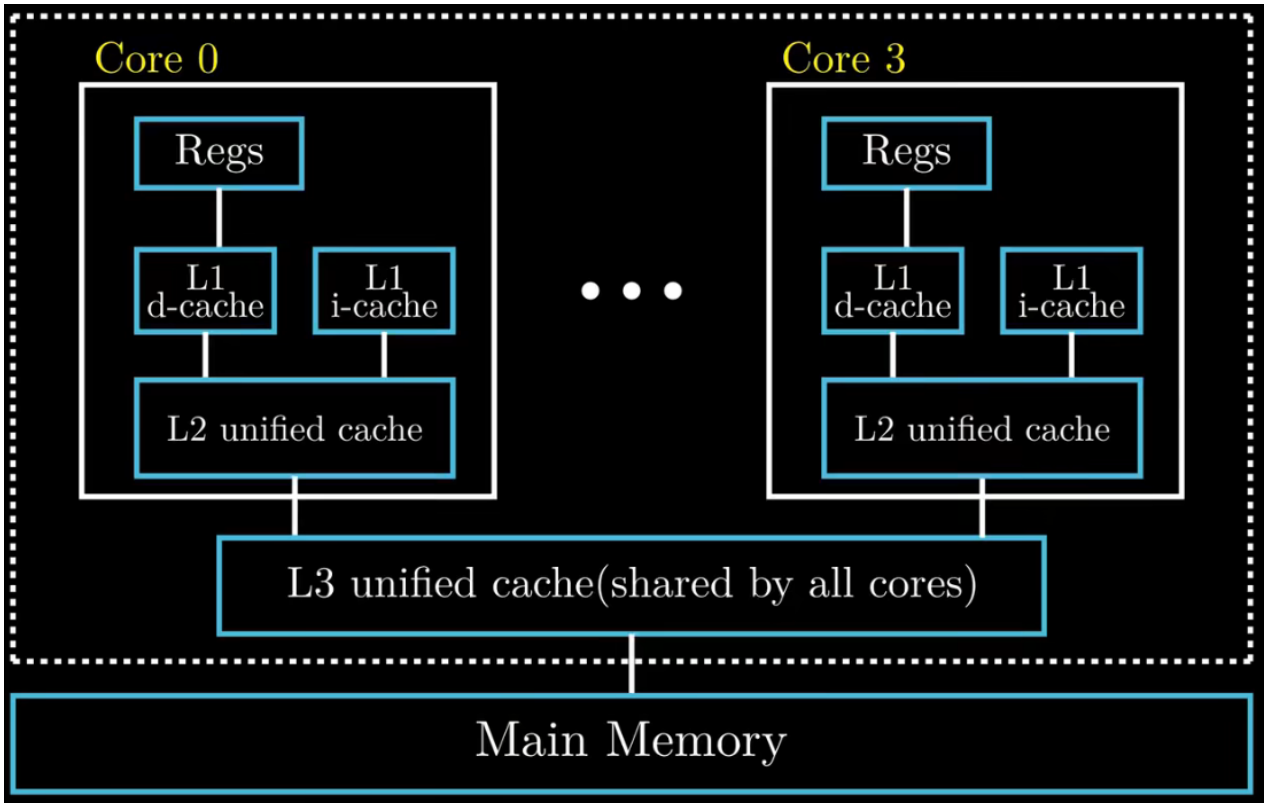
写回write-back：写操作时，信息只写入Cache,而不写入主存，仅当替换时，才将修改过的Cache内容块写回到主存中

写不命中

写分配write-allocate：除了写入主存外，还将该块由主存调入Cache

写不分配no-write-allocate：只写入主存，不将该块从主存调入Cache

示例

> Intel高速缓存层次结构
>
> 
>

Cache的性能评价及影响因素

Cache的主要性能指标:

- 命中率(hit rate)、不命中率(miss rate): 命中、不命中的比率
- 命中时间(hit time): 从高速缓存读出一个字至cpu的时间
- 不命中惩罚(miss penalty): 由于不命中所需额外服务的时间

典型情况:

- L1的命中时间, 典型是1 ~ 2个时钟周期
- L1不命中从L2得到服务的惩罚, 典型是5 ~ 10个周期
- L1不命中从主存得到服务的惩罚, 典型是 25 ~ 100 个周期

影响Cache性能的主要因素:

- 存大小: 大的缓存能提高命中率, 但也会增加命中时间
- 大小: 块大有利于空间局部性, 块小却利于时间局部性
- 相联度、替换策略及写策略也会直接影响Cache的性能

虚拟存储器

> 传统内存管理的问题
>
> - 程序要一次性装入内存
> - 程序的局部性没有利用
> - 内存容易被破坏

交换与覆盖技术

将暂停运行进程的地址空间与外存中由阻塞变为就绪进程的地址空间进行交换

覆盖技术的原理：程序必要部分的代码和数据常驻内存；可选部分存放在外存中，在需要时才装入内存

结果

- 影响程序的结构
- 增加处理机的额外处理时间
- 没考虑程序局部性的影响

原因

- 为进程分配的空间是连续的
- 使用的地址都是物理地址

虚拟地址空间与寻址

虚拟存储器

- 保护进程地址空间的独立性
- 自动管理主存——辅存两级的存储层次

物理寻址：直接从主存中进行寻址

虚拟寻址：CPU从地址控制单元MMU进行虚拟寻址，地址控制单元在通过物理寻址从主存寻址

虚拟内存作为缓存工具

主要作用

- 作为缓存工具提高主存的使用效率
- 扩展程序的内存空间
- 作为管理工具提供系统必需的存储器管理的基本功能.

存储管理方式：段式、页式、段页式

主存作为缓存的结构

页表结构

虚拟页(省略)	有效位	物理页

页表标目：PTE=2^{n-p}，n为虚拟地址大小，页面大小为P=2^{p}

页命中

缺页

页调度策略

按需页面调度策略

预取式调度策略

页面分配-全相联映像

虚拟内存作为内存管理的工具

给进程分配独立的页表, 为进程建立独立的地址空间

简化链接、简化加载、简化共享、简化内存分配

虚拟内存作为内存保护的工具

页式存储

地址翻译

VPN-VPO --> PPN-PPO

已知：虚拟地址空间n位，物理地址m位，页面大小P

- VPO=PPO=$\log_2 P$

- VPN=n-VPO
- PPN=m-PPO

CPU处理步骤

TLB快表优化

多级页表

异常控制流

程序控制流

CPU执行的地址序列称为控制流，顺序执行或者跳转这种与程序的执行相关的方式得到的控制流称为正常控制流

程序的执行过程中碰到了意外的情况，系统需要相应的机制去处理意外的情况，该类情况称为异常控制流

异常控制流（Exceptional Control Flow，ECF）

定义：由于某些特殊情况引起用户程序的正常执行被打断，所形成的意外控制流称为异常控制流

理解ECF的重要性：

- 操作系统的重要功能
- 应用程序与操作系统的交互
- 编写程序
- 并发程序的执行

异常控制流存在于计算机的各个层面

异常

异常是操作系统为了响应一些事件而做出的控制流的改变。就是CPU遇到了一些特殊的情况，使得正在执行的程序被终止，转到处理异常的情况，结束之后再返回被终止的程序。

异常表

异常类别

异步异常：由当前程序之外的因素引起的异常

- 中断

同步异常：由当前程序本身引起的异常

- 陷阱
- 故障
- 终止

中断

引起原因：

- 定时器触发
- 外部设备的输入输出

中断一定发生在一条指令结束后，返回到当前指令的下一条

陷阱和系统调用

陷阱是专门实现的异常，它发生在程序内部，是预先安排的事件，如单步跟踪、断点等。

系统调用：利用陷阱提供了一种机制用来实现用户程序和内核之间的访问接口。

故障

故障由错误情况引起。但是可以被处理程序修正。但是修正之需要重新执行这条引起故障的指令

故障发生时，处理器将控制转移给故障处理程序，如果处理程序能够修正错误，就把控制返回到引起故障的指令，否则返回给内核中的 abort 例程，abort 会终止当前的应用程序。

终止

也是由错误引起的，但是这种错误不能修复而且不能确定错误是哪条指令产生的，一旦出现终止，控制将会给Abort的例程，中止当前的应用程序

事件	异常
硬件电路故障	终止
数据溢出	故障
缺页	故障
非法指令	终止
除数为零	
断点设置	

系统调用

每个系统调用都有一个唯一的ID号

异常指令处理

进程

逻辑控制流

每个进程都是一个逻辑控制流

并发&顺序：如果两个逻辑流在时间上有重叠，则称是并发的，否则是顺序的

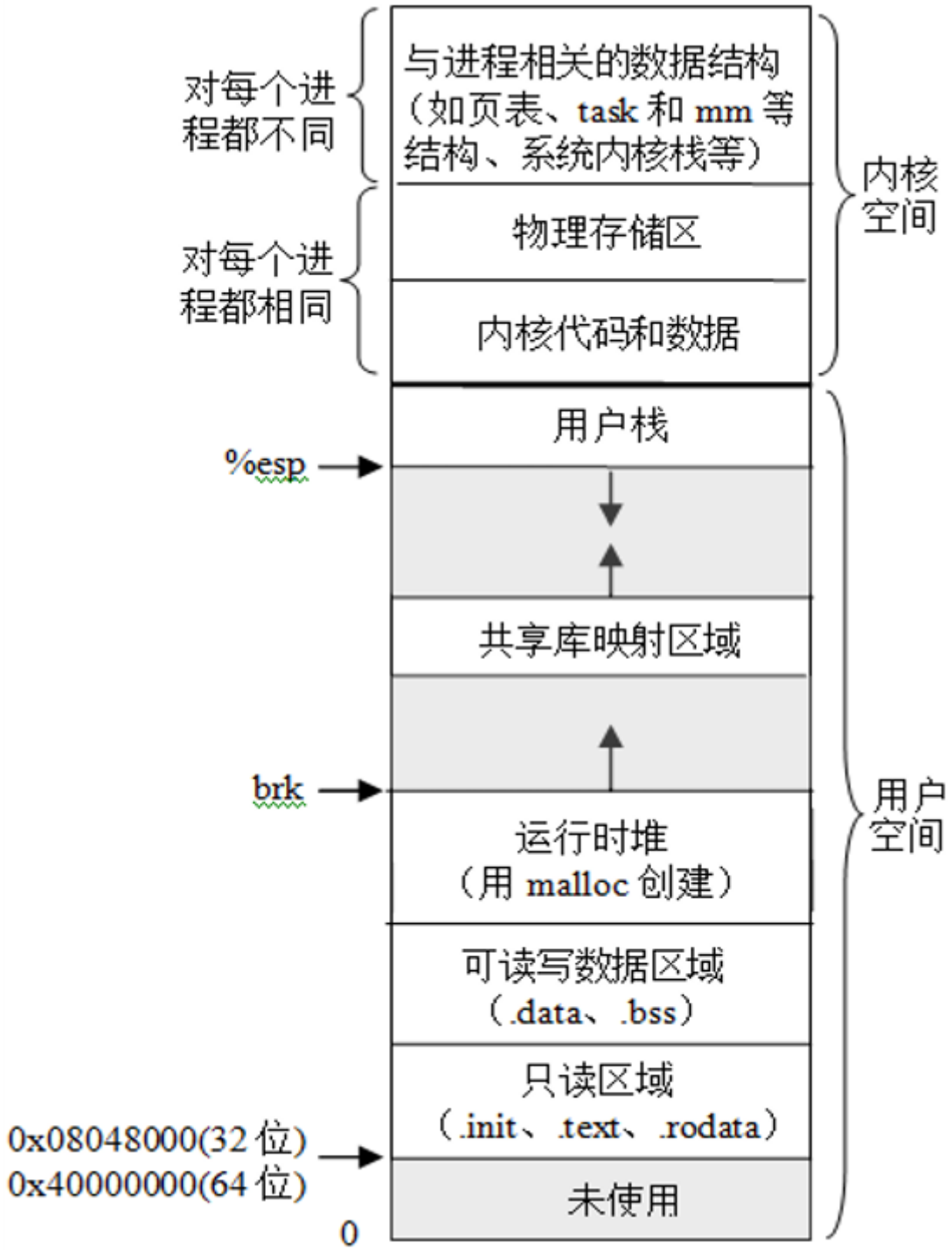
并发流：一个逻辑流在执行时间上与另一个流重叠，称为并发流

多任务：一个进程与其他进程轮流运行称为多任务

时间分片：一个进程执行它的控制流的一部分的每一时间段称为时间片，所以多任务称为时间分片

并行流：如果两个流并发地运行在不同的处理机或者是处理器核心上，就称之为并行流

私有地址空间：Linux平台每个进程都有独立的私有地址空间(虚拟地址空间)，每个进程的地址空间划分布局相同



内核模式

内核空间存放的是操作系统内核代码和数据，是被所有程序共享的，在程序中修改内核空间中的数据不仅会影响操作系统本身的稳定性，还会影响其他程序，所以操作系统禁止用户程序直接访问内核空间

用户模式

为什么内核和用户要共用地址空间

发生系统调用时进行的是模式切换，模式切换仅仅需要寄存器进栈出栈，不会导致缓存失效

上下文切换

每个任务运行前，CPU 都需要知道任务从哪里加载、又从哪里开始运行，这就涉及到 CPU 寄存器 和程序计数器（PC）

CPU 寄存器是 CPU 内置的容量小、但速度极快的内存；

程序计数器会存储 CPU 正在执行的指令位置，或者即将执行的指令位置。

这两个是 CPU 运行任何任务前都必须依赖的环境，因此叫做 CPU 上下文