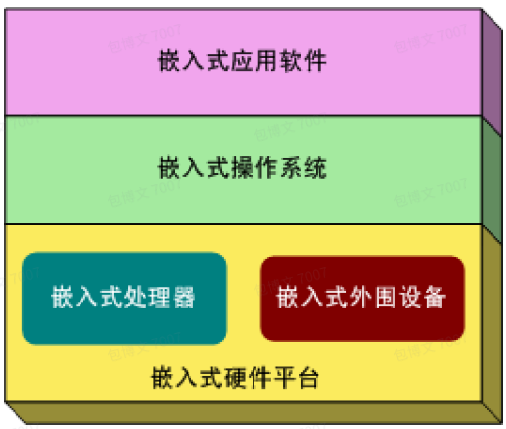
最后一课

1 导论

- **嵌入式系统**是以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。
- **嵌入式系统的特点**：面向应用；关注成本；对实时性、可靠性有要求；交叉开发；
- **嵌入式系统与通用计算机的对比区别**

特征	通用计算机	嵌入式系统
形式和类型	看得见的计算机。按其体系结构、运算速度和结构规模等因素分为大、中、小型机和微机	看不见的计算机。形式域广泛，按应用来分。
组成	通用处理器、标准总线和外设。软件和硬件相对独立。	面向应用的嵌入式微处外部接口多集成在处理与硬件是紧密集成在一
开发方式	开发平台和运行平台都是通用计算机	采用交叉开发方式，开通用计算机，运行平台
二次开发性	应用程序可重新编制	一般不能再编程

- **嵌入式系统的组成（不需要背分类）**



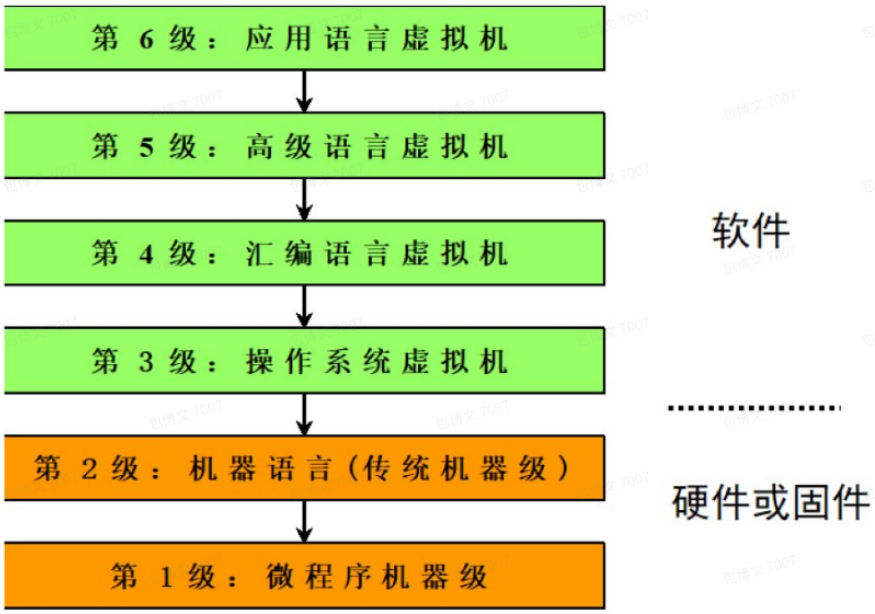
2 处理器设计导论

- **体系结构概念：**体系结构描述从用户角度看到的计算机，即概念结构与功能特性。

计算机系统 = 软件+硬件/固件，计算机语言由低级向高级发展（高一级语言的语句相对于低级语言功能更强，更便于应用，但又都以低级语言为基础）。从计算机语言的角度，把计算机系统按功能划分成多级层次结构。

概念的实质：计算机系统中软硬件界面的确定，其界面之上的是软件的功能，界面之下的是硬件和固件的功能。

体系结构的核心：指令集结构



- **组织概念**：计算机组织描述从用户角度不能看到的体系结构的实现方式:

  流水线结构(Pipeline)

  高速缓存 (Cache)

  步行表硬件 (table-walking)

  转换后备缓冲 (TLB)

- **实时操作系统（RTOS）**是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应，并控制所有实时任务协调一致运行的操作系统。因而，提供及时响应和高可靠性是其主要特点。

- **什么是硬实时和软实时，试举例说明。**

硬实时：必须满足最后期限的限制，否则会给系统带来不可接受的破坏或致命错误

软实时：有一个与之关联的期限，并希望能够得到满足，但不是强制的。即使超过这个期限，调度和完成这个任务仍然有意义。

- 指令集设计
  - arm寻址模式：立即寻址，寄存器寻址，寄存器间接寻址.....。arm一开始没有直接寻址，因为地址和指令都是32bit

- RISC
  - UCB RISC I/II
  - STF MIPS

3 ARM体系结构

- **简述ARM体系结构的主要特点、ARM继承RISC**

Load/Store结构

3地址的数据处理指令前三者为RISC的继承

强大的多寄存器Load/Store指令

32位指令

单时钟周期，单条指令完成一项普通的移位操作和一项普通的ALU操作

所有指令条件执行

通过协处理器指令集扩展ARM指令集，在编程模型中增加了新的寄存器和数据类型

- 版本(ARM Archi v X-> ARM Processor X)
  - ARMv4T:在THUMB体系结构中以高密度的16位压缩形式表示的指令集
  - TE: Enhanced DSP
  - TEJ: Java
  - v7: Profile A, R, M
- arm继承RISC
  - Load/Store
  - 32bit instruction
  - 3 address instruction format
  - 有的未采用

4 ARM编程模型

- 存储
  - Byte， Half-word, Word的访问， “对其”
  - Little/ Big Endian: Big LSB 0x 12 34 56 78 MSB
- **ARM处理器模式**

处理器模式	说明	备注

用户(usr)	正常程序工作模式	**不能直接切换到其它模式**
系统(sys)	用于支持操作系统的特权任务等	与用户模式类似，但具有可以直接切换到其它模式等特权，与用户模式共用寄存器，此模式修改寄存器更方便
快中断(fiq)	支持高速数据传输及通道处理	FIQ异常响应时进入此模式
中断(irq)	用于通用中断处理	IRQ异常响应时进入此模式
管理(svc)	操作系统保护代码	**系统复位和软件中断**响应时进入此模式
中止(abt)	用于支持虚拟内存和/或存储器保护	在ARM7TDMI没有大用处
未定义 (und)	支持硬件协处理器的软件仿真	未定义指令异常响应时进入此模式

- **处理器启动过程**

  管理模式 `svc` （复位后默认）——多种特权模式（主要设置各模式堆栈）——用户程序运行模式

- **寄存器描述**

  ARM有37（31+6）个物理寄存器，有18个可编程访问的寄存器；

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

- **ARM状态寄存器**

CPSR程序状态寄存器： 4个条件代码标志（最高四位）+ 最低八位（0-7）为控制位，发生异常时被硬件改变或特权模式下软件改变（就那种允不允许fiq、允不允许中断啥的，因为不同异常有优先级嘛）

通通 不需要大家去背

异常类型	优先级
复位	1（最高优先级）
数据中止	2
FIQ	3
IRQ	4
预取中止	5
未定义指令	6
SWI	7（最低优先级）

- R14

- **异常、异常进入与退出**

正常的程序流被暂时中止，处理器就进入异常模式。

进入异常

1.在对应模式的LR中保存下一条指令的地址

异常入口为当前指令+8/4（ARM状态）或当前指令+2/4（Thumb状态）

2.将CPSR复制到对应SPSR（程序状态保存寄存器）中；

3.强制设置CPSR处理器模式位为与异常类型相对应的值；

4.强制PC从相关的异常向量出取址（进入异常）。

IRQ和FIQ应置位禁止标志，防止不受控制的异常嵌套。

退出异常

1.将LR中的值减去偏移量后存入PC，偏移量根据异常的类型而有所不同；

2.将SPSR的值复制回CPSR；

3.清零在入口置位的中断禁止标志。

恢复CPSR的动作会将T（状态位）、F和I位（禁止标识）自动恢复为异常发生前的值。

- **ADD R3,R2,R1,LSL#3能否一周期完成？理由**

  可以。本题先移位，再加法，移位指令与ALU操作合并在一个指令周期里进行。

- **ARM指令集按照功能划分，可以分为以下类型：**数据处理、数据传送、控制流指令
- **寻址模式：**立即寻址、绝对寻址（ARM不行）、间接寻址、寄存器寻址、寄存器间接寻址、基址偏移寻址、基址变址寻址、基址比例变址寻址、堆栈寻址、块拷贝寻址
- **数据传送指令**

  单寄存器Load/Store指令支持的寻址方式：寄存器间接寻址⊠基址偏移寻址⊠基址变址寻址

  多寄存器Load/Store指令两种最有用的寻址方式：1堆栈寻址（堆栈是一种后进先出的存储形式，支持动态存储器分配；2块拷贝寻址（把一个数据块从存储器一个位置拷贝到另一个位置）

- **控制流指令可能的形式**

  使指令执行切换到不同的地址

  1 ⊠永久转移——转移指令（B）：使处理器由顺序执行指令转移到Label处，利用的寻址方式是PC相对寻址。

  2 ⊠保存返回地址以恢复原来的执行顺序（BL）

  3 ⊠软件中断指令（SWI）：处理器进入管理（监控）模式

5 体系结构对高级语言的支持

- **ARM体系结构对C数据类型的支持（**选择，应该不会简答，简答太逆天了）：

有符号和无符号的32位整数——字边界对齐

有符号和无符号的16位短整数——偶数地址对齐对齐

8位的无符号字节——字节对齐

8位的有符号字节——字节对齐

枚举与位域类型通过整数实现

指针通过ARM本身的地址实现(32位)

- ARM 体系结构对 C 数据类型的支持：
  - 对数组的支持—使用基址比例变址的寻址模式
  - 对结构的支持—使用基址立即数偏移寻址模式
  - 通过 FPA10 硬件浮点加速器和 VFP 浮点协处理器提供很好的支持

- **有关ATPCS规范的描述（不定项）**

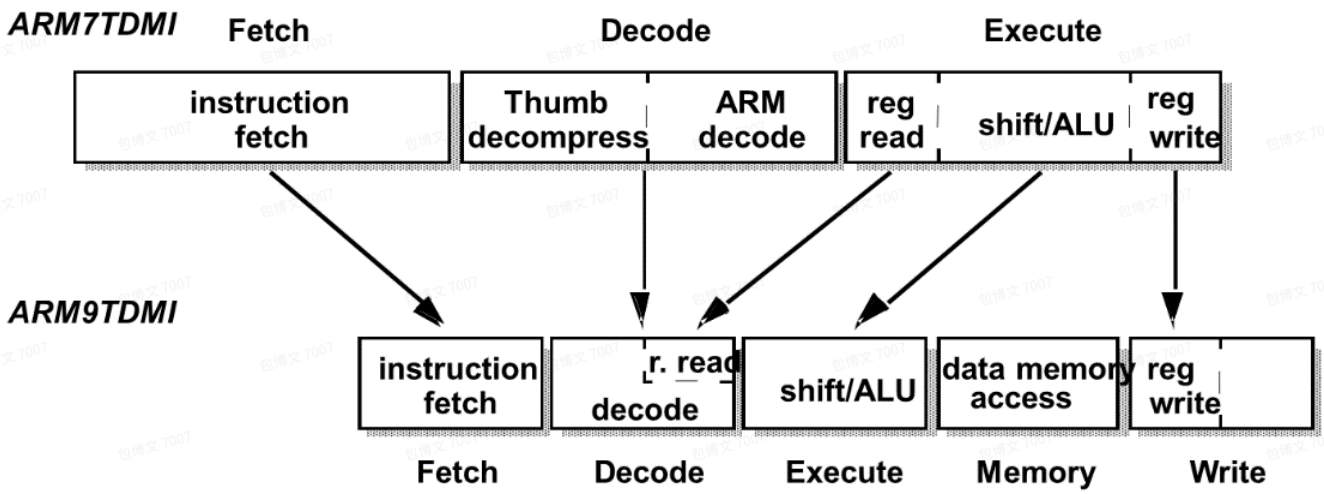
  ARM THUMB Procedure Call Standard子程序间调用的基本规则

（寄存器规则、数据栈规则 Full Desending、参数传递规则，展开部分太多了感觉没必要记）

- Stack, heap

6 流水线组织（简答很多）

- 为什么要有流水线
- 多周期指令流水线
- 用图表示出三级流水线与五级流水线对应关系



- **ARM 7、ARM 8 、ARM9 、ARM10流水线比较和对应关系（7&8用图表示应该就是三级流水线和五级流水线对比图）分析提高处理器性能的方法（提高处理器性能的方法考很多次，重要）**

$$T_{prog}=N_{inst}\times CPI/F_{clk}$$

- N：指令数
- CPI：每条指令的平均时钟周期数
- F：处理器的时钟频率

减小指令数量：优化编译器

提高时钟频率表：增加流水线级数，从而简化每一级的逻辑（即减少每个时钟周期内必须完成的最大工作量）

减小CPI：

根本问题——存储器瓶颈（指令和数据在同一存储器，性能受到存储器带宽限制）

改进方法：

- 一个时钟周期给出多于32位数据、哈佛结构（指令、数据分开存储）
- 增加级数后⊠可能产生后一条指令需要前一条指令的结果，然而前一条指令结果还未回写至寄存器，因此需要前推通路，从而可以使得结果能在级间传送。

具体案例分析：

ARM8相对于ARM7：

增加时钟速率：简化每级流水线的逻辑，增加级数，**五级流水线**

降低CPI：将ARM7占据一个以上流水线槽的指令重新实现，以占据较少的流水线槽，减小由指令间的相关性引起的流水线停止

ARM9相对于ARM8：

降低CPI：使用分开的指令与数据存储器端口改善CPI（**哈佛结构**）

硬件直接译码ARM指令核Thumb指令进行，流水线内没有时间对其进行软译码

可以在异常时设置断点

ARM9相对于ARM8：

两倍性能

增加时钟速率：采用特别方式优化每一级，在译码前**增加发射一级**

降低CPI：主要改进来源：**存储器带宽**采用64位存储器

7 存储器层次与高速缓存&外设

SRAM

SDRAM

层次	名称	容量	速度
1	寄存器	ARM 包含 32 个 32 位寄存器，共 128B	几个 ns
2	L1 Cache	8 — 32KB	10ns
3	L2 Cache	几百 KB	几十 ns
4	主存储器	几 MB — 几十 MB	100ns
5	后援存储器	几十 MB — 几百 GB	几十 ms

寄存器中的数据可以由编译器或者汇编语言直接控制，其他层次中的内容通常为自动管理；Cache对于应用程序往往是不可见的，指令和数据以块或页的形式移动；

- Cache命中（hit）：
  - CPU每次读取主存时，Cache控制器都要检查CPU送出的地址，判断CPU要读取的数据是否在Cache中，如果在就称为命中。命中率90%+

主存储器与后援存储器的映射由操作系统控制，对于应用程序是透明的。

- **CACHE分类**：
  - 数据Cache、指令Cache
  - 统一Cache
- **CACHE策略：**

属性	物理缓存（Physical Cache）	虚拟缓存（Virtual Cache）
缓存内容	统一指令和数据缓存	分离指令和数据缓存
关联性	直接映射	组相联
替换策略	循环	随机
写策略	写直达（Write-through）	写直达（带写缓冲，Write-through with write buffer）
完全关联性	-	CAM-RAM
替换策略（完全关联）	-	最近最少使用（LRU）
写策略（完全关联）	-	写回（Copy-back）

缓存替换策略有最近不经常使用(LRU)、先进先出(FIFO)等，用来确定替换哪些数据；

而缓存写策略主要有写回(Write Back)和写直达(Write Through)，用来决定何时将数据写回主内存。

- **RAM-cache对比**：

  RAM简单、便宜且功耗低

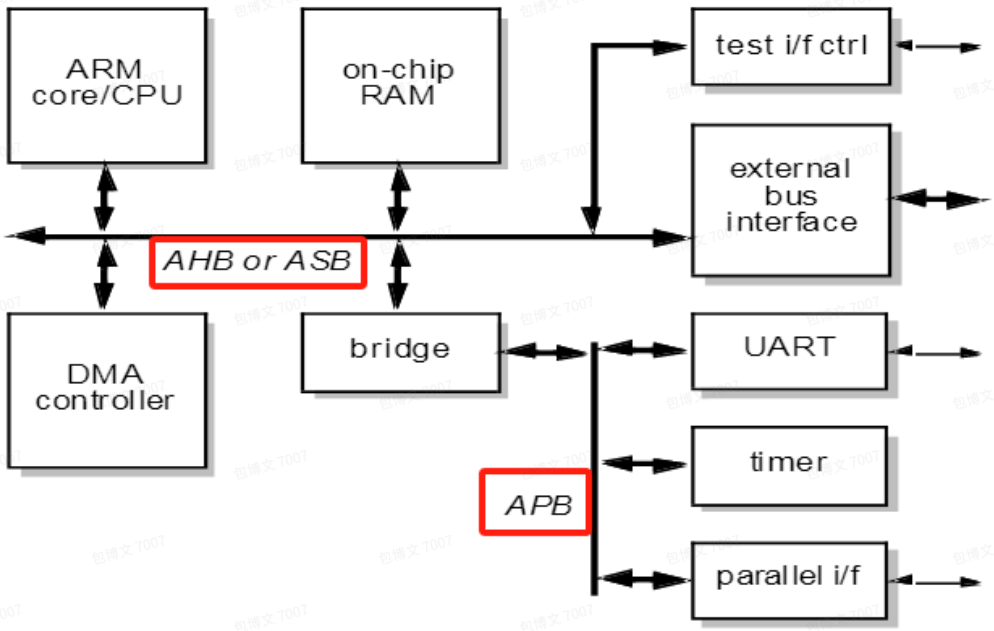
  有更确定的行为

  使程序员能够根据对将来处理工作量的了解来划分 RAM 空间

  缺点是需要程序员直接管理，不是透明的，不易于程序的混合

- **AMBA三总线**

  AHB（high、总单元、从单元、仲裁器、译码器）、ASB（system）、APB（Peripheral、本地二级总线）



- **外设如UART的工作原理，arm如何驱动**

1串口初始化：读写各种控制寄存器设置串口奇偶校验位、停止位、串口数据位长度；使能串口发送和接收；串口波特率

2发送数据：控制寄存器中的发送位置为“1”，开始发送。FIFO使能就直接发，没使能就存在它下面一个字的位置。

3 接收数据：查询接收状态寄存器，判断是否有数据来，来了就读取。FIFO使能就直接读，没使能就存在它下面一个字的位置。

8 存储器管理单元 & JTAG测试与调试结构

- **存储器管理的两种基本方法：**段式管理（segmentation）、页式管理（paging ARM用）
- 在分页式存储管理中，地址转换工作是有（**逻辑地址（也叫虚拟地址）到物理地址**）引起的。
- 分页管理每取一数据，要访问（**2**）次内存。

- MMU功能
  - **MMU(存储器管理单元)实现ARM处理器虚拟地址到物理地址的转换，MMU转换时可能发生的故障**：对齐、转换、页域以及权限故障
  - **MMU通过怎样的方式禁止对ARM的非法访问？**设置访问权限
- **存储器粒度：**段（1MB），大页（64KB）、小页（4KB）、微页（1KB）
- JTAG：Joint Test Action Group Boundary Scan
- **JTAG信号组成：**（四输入一输出）带时钟串口 时钟，模式，复位，输入，输出

  TCK（独立于系统时钟的测试时钟）

  TMS（测试模式选择信号）

  TDI（测试数据输入）

  TRST（测试复位输入）

  TDO（测试数据输出）
- **JTAG边界扫描测试支持的最小集公共指令包括：**

  BYPASS：器件将TDI经过1个时钟延时连接到TDO，用于同一个测试环中其他器件的测试

  EXTEST：将边界扫描寄存器连接到TDI和TDO之间，能够捕获核控制引脚状态。这条指令用于支持板级连接测试

  IDCODE：将ID寄存器连接到TDI和TDO之间

  INTEST：将边界扫描寄存器连接到TDI和TDO之间，能够捕获和控制核逻辑的输入及输出状态。这条指令用于内部逻辑核的测试
- **宏单元概念：**

☒系统芯片使用大量的复杂、已设计好的宏单元（如ARM处理器核本身）

☒宏单元的产品测试向量主要依赖于宏单元供应商
- **将测试向量加到宏单元的方法：**

☒通过多路器使每个宏单元的信号依次连接到系统芯片的引脚上的测试模式

☒片上总线（AMBA总线）可以支持每个连接到总线上的宏单元的直接测试访问

☒每个宏单元可以有一个边界扫描路径，使用扩展的JTAG结构，测试向量可以通过扫描路径加到宏单元上（同样有速度过低的缺点）
- **PCB**：支持JTAG的用EXTEST指令，不支持的用探针技术
- **桌面调试**

  调试器与要调试的系统都是运行在同一台PC上的不同程序，所有用户接口准备好，采用断点和调试符号表，普遍缺乏观察点工具
- **嵌入式系统调试**

调试工具必须在远程机上运行，并通过某种通信方式与目标机器连接。通常采取在线仿真器（ICE，通常为一个相同的芯片或更多引脚的变型芯片）替代目标系统中的处理器，并包含缓冲器以实现硬件跟踪。

几种常见的调试方法：

1指令集模拟器：一种利用PC机端的仿真开发软件模拟调试的方法。

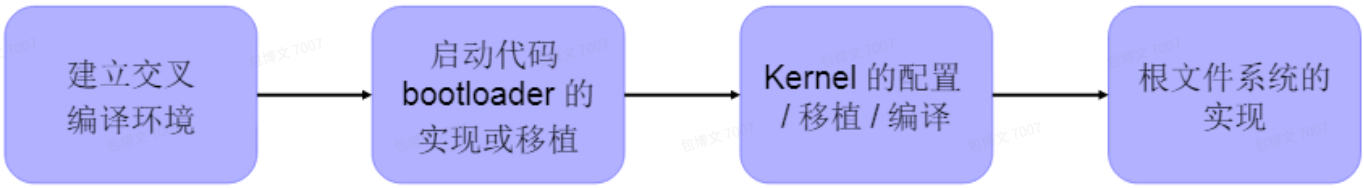
2驻留监控软件：驻留监控程序运行在目标板上，PC机端调试软件可通过并口、串口、网口与之交互，以完成程序执行、存储器及寄存器读写、断点设置等任务

3 JTAG仿真器：通过ARM芯片的JTAG边界扫描口与ARM核进行通信，不占用目标板的资源，是目前使用最广泛的调试手段

4在线仿真器：使用仿真头代替目标板上的CPU，可以完全仿真ARM芯片的行为。但结构较复杂，价格昂贵，通常用于ARM硬件开发中

9 嵌入式系统开发基础

- **嵌入式系统 软件开发**



- **嵌入式（软件）系统的启动步骤 要了解**

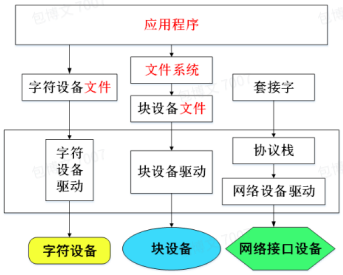
X-loader：简化的bootloader配置片外SDRAM，并 将bootloader引导至片外SDRAM中执行(板级外设、 串口控制台初始化)

10 Linux设备驱动基本原理

- **Linux设备类型：**

驱使硬件设备行动，负责软硬件之间的沟通。提供机制，而不是策略。

Linux系统将设备分成三种基本类型， 每个模块通常实现其中的某一类：字符模块、块模块和网络模块 。相对应的，设备驱动程序也可分为字符设备驱动程序、块设备驱动程序、网络接口驱动程序。



11 进程

- **进程**是一个活动的单元，它有一个执行语句序列（顺序串，就称为轨迹Trace），一组当前状态和一组相关的系统指令。包括（程序代码、数据、描述进程状态的属性）
- **进程控制块PCB**：记录进程的相关的信息，由OS创建并管理，支持多个进程。

  –进程标志–进程状态 –进程优先级–程序计数器–内存指针–上下文数据–I/O 状态信息–统计信息
- **进程创建条件**

进程的诞生与终结

创建	终结
新的批处理任务	正常结束
交互登陆	内存不可用
由 OS 建立提供服务	保护出错
由现有进程派生	OS 或操作员干涉

- **进程派生**：让正在运行的进程可以建立进程，会很有用
- **进程挂起**：处理器会比I/O快很多，所以所有进程都会碰到等待I/O返回的问题——把这些等待的进程交换到磁盘，可以释放出内存并使处理器可以处理其它更多的进程，当swap到磁盘时，进程阻塞的状态转为挂起(suspend)

进程挂起的原因

Reason	Comment
内存交换	OS 需要释放内存以运行其它 ready 状态的进程
其它 OS 原因	OS 因为出错而挂起了进程
交互用户的请求	比如调试过程中，或者与某个资源的使用有关
分时处理	OS 可能会调度一个进程运行一段时间，然后将它暂停，并把处理器转给其它进程 .
父进程请求	父进程可能会希望将它创建的子进程暂停一会儿，以检查或修改进程状态，或者对多个子进程进行协调 .

- **操作系统控制结构**
  - 内存表、
  - I/O表、
  - 文件表、
  - 进程表（当前状态、进程ID (PID)、内存中位置、etc）

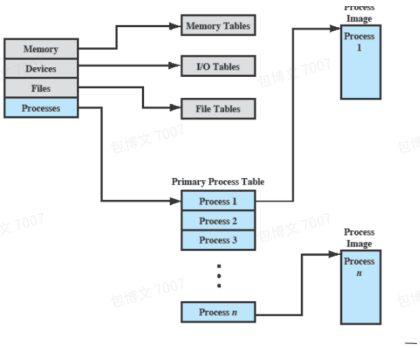


Figure 3.11 General Structure of Operating System Control Tables

- **运行模式**：用户模式&系统模式

- **创建进程：**一旦OS决定创建一个新进程，它将：
  - 分配一个独一无二的PID
  - 给进程分配内存空间
  - 初始化PCB
  - 设置相应的链接
  - 建立其它的数据结构
- **切换进程：**在OS从当前进程获得控制权时，就会发生进程切换

  触发机制：中断、陷阱、管理调用，

  步骤：

  1.保存处理器的上下文(context)，包括PC和各寄存器

  2.更新进程的PCB，此时它本是Running状态， 修改为其它状态(Ready; Blocked; Ready/Suspend; or Exit)....

  3.将此PCB移至相应的队列中 –ready; blocked; ready/suspend

  4.选择另一个需要运行的进程

  5.更新被 选中的进程的PCB，包括修改状态为Running

  6.更新内存管理相关的数据结构，不同的地址转换机制需要不同的处理

  7.恢复被选中进程相关的context，包括PC 指针和寄存器，使进程恢复到上次被切出Running状态前的情形
- **进程上下文：**进程执行全过程的静态描述。需要提供给操作系统的相关信息。

12 互斥与同步

- **临界区：**两个进程或线程访问一个共享变量时，会产生临界区问题。
- **竞争条件：**多个进程或线程读或写数据项；各进程执行的顺序，会影响最终结果；输出依赖于谁最后完成执行流程
- **互斥的要求：**1对于资源的临界区，同一时间只有一个进程可以访问；2进程在非临界区被挂起时，必须不可干扰其它进程；3不出现死锁或饿死；4进程在其临界区没有被其它进程占用时不应被延迟；5不应对进程运行的速度或进程个数作假设；6进程在其临界区仅停留有限时间。
- **Producer/Consumer问题：**

  总体情况：一个或多个 producers 产生数据，并将它放到一个buffer；一个 consumer 从buffer中取出数据，每次一个；在同一时间只能有一个producer 或 consumer访问 buffer

  问题:确保Producer不会在buffer满时添加数据，而consumer 不会在buffer空时移出数据

- **Readers/Writers 问题：**

  一个数据区，为多个进程共享，有些进程只对数据区作读操作，有些只写。

  要满足的条件：

  1.多个readers可能同时读一个文件

  2.同一时刻，只能有一个 writer可执行写操作

  3.如果writer正在写入文件期间，reader不能读

13 死锁与饥饿

- **死锁：**一组进程，当其中每一个进程都因等待某一事件而被阻塞，而这事件仅可由其它进程触发，则发生死锁。典型的情况，是进程对同一资源的竞争
- **死锁发生的必要条件：**互斥、持有并等待、不可剥夺、循环等待
- **死锁避免（哲学家问题）：**解决方法1只允许四个哲学家同时吃，2让哲学家4取叉子顺序与其它哲学家不同–先取fork[0]，后取fork[4]，3 AND同步，只要任一个信号量不能获得，就一个也得不到，调用进程被阻塞
- **银行家算法**

树莓派开发

- **谈谈你对树莓派开发的认识和感受**