**嵌入式Linux 期末复习**

1. **交叉开发、交叉开发环境？为何需要交叉开发环境？**

答：在一台通用计算机（宿主机）上进行软件的编辑编译，然后下载到嵌入式设备（目标机）中运行调试的开发方式，交叉开发环境一般由运行于宿主机上的交叉开发软件、宿主机到目标机的调试通道组成。

需要交叉开发环境是因为目标机一般对体积、功耗等有严格限制，资源也面向应用，较为紧

张，要求仅仅能流畅运行代码即可，而将用户开发软件（包括各种库、工具）放置在主机上，

而且现在的集成开发环境提供了各种修改好的功能库，用起来也方便。

**2.1 什么是根文件系统？Linux内核启动与根文件系统的关系？**

答：根文件系统是Linux系统的安装点，应该包括支持Linux系统正常运行的基本内容，包含系统使用的软件和库，以及所有用来为用户提供构架和用户使用的应用软件。Linux内核在运行期必须要用到的存储代码体系，与内核同时存在，只在内核运行时才存于内存中。根文件系统必须能够提供Linux内核启动过程中要加载的模块。

**2.2 根文件系统**

文件系统就是把你硬盘上数据按照一定格式组织成一棵树。数据块对应名称。删了它就相当

于把硬盘格式化了。

根文件系统就是出了内核以外，所有的系统文件存储的地方。之所以成为根，是因为有根才

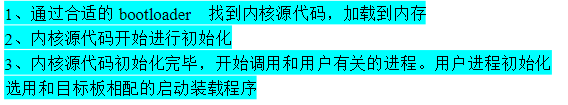
能成生长成树，是其它文件的最终挂载点。

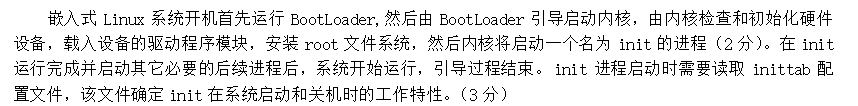
我们要明白 根 文件系统和内核是完全独立的两个部分，它是内核启动时所 mount 的第一个文件系统，里面有内核启动所必须的数据，不然就退出启动。

文件系统这种机制有利于用户和操作系统的交互。数据块对应名称。尽管内核是 Linux 的

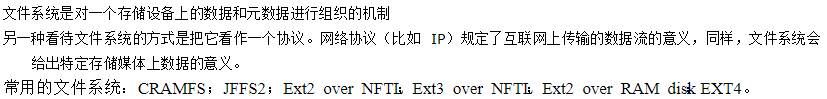
核心，但文件却是用户与操作系统交互所采用的主要工具，尤其是 LINUX。

1. **Linux内核在目标板上启动的过程？需要考虑哪些因素**





1. **如何理解根文件系统？常用的文件系统**



1. **嵌入式 Linux 开发主要流程？**

**搭建开发环境--烧写 bootloader--烧写内核--烧写根文件系统--烧写应用程序。**

开发环境：REDHAT－LINUX、下载相应的 GCC 交叉编译器进行安装、 配置开发主机（配

置 置 MINICOM 和配置网络，MINICOM 软件的作用是作为调试嵌入式开发板信息输出的监视器和键盘输入的工具，配置网络主要是配置 IP 地址、NFS 网络文件系统，需要关闭防火墙）

**烧写 bootloader**

下载一些公开源代码的 BOOTLOADER 根据自己具体芯片进行移植修改。

下载时，

有些芯片没有内置引导装载程序，比如三星的 ARM7、ARM9 系列芯片，这样就需要编写

烧写开发板上 flash 的烧写程序。

或者网络上有免费下载的 WINDOWS 下通过 JTAG 并口简易仿真器烧写 ARM 外围 flash

芯片的程序。也有 LINUX 下公开源代码的 J-FLASH 程序。

**下载内核**

如果有专门针对你所使用的 CPU 移植好的 LINUX 操作系统那是再好不过，下载后再添加

自己的特定硬件的驱动程序，进行调试修改。

**下载根文件系统**

从 www.busybox.net 下载使用 BUSYBOX 软件进行功能裁减，产生一个最基本的根文件系

统。根文件系统在嵌入式系统中一般设为只读，需要使用 mkcramfs 、genromfs 等工具产生烧写映象文件。

**下载用户程序**

可以下载到根文件系统中，有的应用程序不使用根文件系统，而是直接将应用程序和内核设计在一起，这有点类似于 UCOS-II 的方式

1. **什么是 bootloader**

Bootloader，为引导加载程序，是嵌入式系统加电后运行的第一段代码，相当于PC 机的BIOS。

Bootloader 在系统中的位置通常固化在硬件上的某个固态存储设备上，加电后自启动。

Bootloader 功能

初始化，给CPU 合适的工作环境（相当于 STARTUP.S），以便为最终调用操作系统内核或

用户应用程序境。

加载内核

下载内核或者根文件系统。

Bootloader 操作模式

有启动加载和下载两种模式。

启动加载模式是 Bootloader 的正常工作模式，在嵌入式产品发布的时侯，Bootloader 必须工作在这种模式下。即初始化 CPU 的工作环境之后，将内核如 RAM 执行。

下载模式：

目标机上的 Bootloader 将通过串口连接或网络连接等通信手段从主机下载文件。

主要是下载内核映像和根文件系统映像等。从主机下载的文件通常首先被 Bootloader 保存到目标机的 RAM 中，然后再被 Bootloader 写到目标机上的 FLASH 类固态存储设备中。

Bootloader 的这种模式通常在第一次安装内核与根文件系统时被使用；此外，以后的系统更

新也会使用到这种工作模式。

Bootloader 启动过程

上电之后，先启动 CPU 即执行 startup.s 类似功能代码（配置中断、初始化堆栈、拷贝代码等），

然后进行加载内核的准备

1、至少初始化一个串口，以便向终端用户反馈数据。

2、检测系统内存映射，哪些是可用的 RAM？

在这一步之后，将检测外部按键，有按键按下将进入下载模式，没有按键的话将执行下面的

步骤，加载内核：

3、将 kenel 和根文件系统从 flash 调入 RAM

4、为内核启动设置参数

5、调用内核。

1. **嵌入式软件开发工作**

初始化引导代码 bootloader编写

嵌入式操作系统代码 内核编译加载

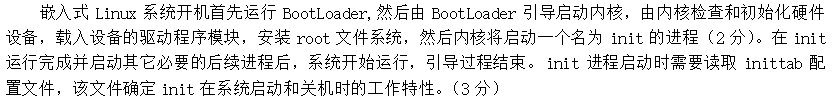
驱动程序代码 驱动模块编写编译

文件系统 文件系统制作挂载

GUI（可选）

应用程序代码 应用程序编写，交叉编译环境构建

1. **嵌入式系统的初始化过程**



1. **进程调度**

进程同程序的比较

程序是指令的有序集合，其本身没有任何运行的含义，是一个静态的概念。而进程是程序在处理机上的一次执行过程，它是一个动态的概念。

程序可以作为一种软件资料长期存在，而进程是有一定生命期的。程序是永久的，进程是暂时的。

进程更能真实地描述并发，而程序不能

进程是由PCB、程序段和数据段三部分组成的

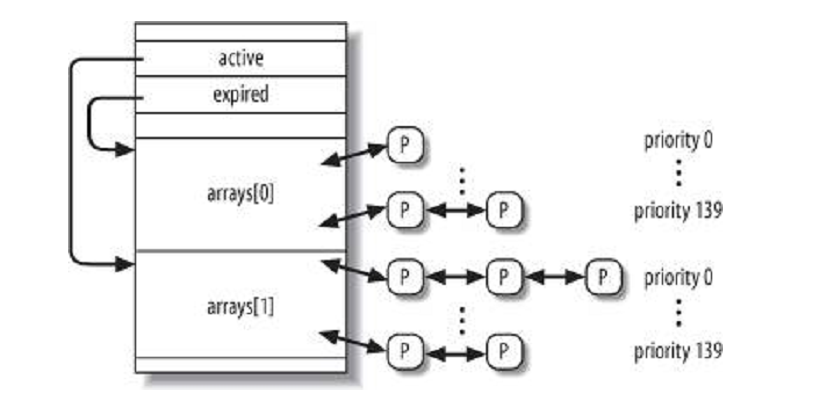
进程具有创建其他进程的功能，而程序没有

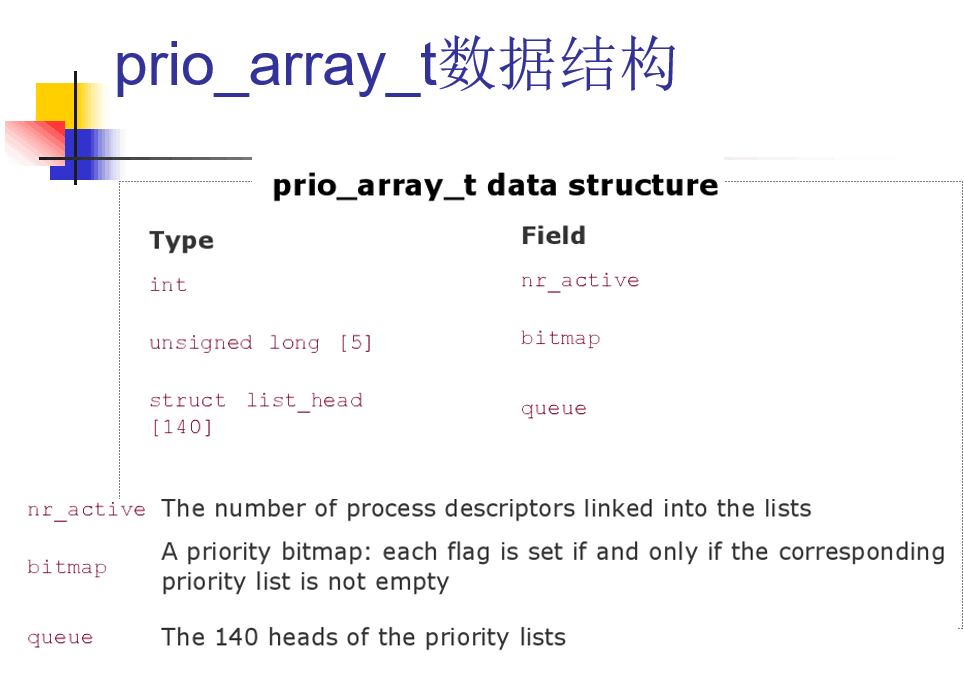
同一程序同时运行于若干个数据集合上，它将属于若干个不同的进程。也就是说同一程序可以对应多个进程

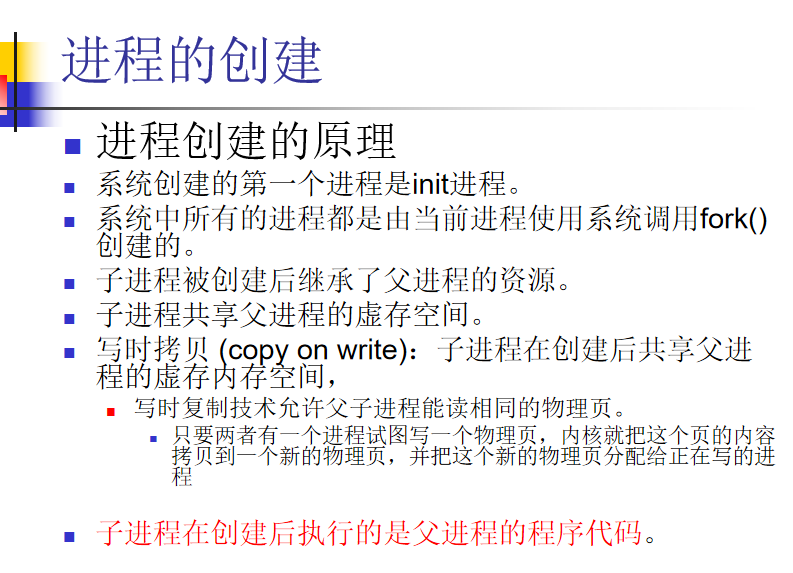
进程特征：结构特征 动态性——最基本特征 并发性 独立性 异步性

进程是由代码(text)、用户数据段(user segment)和系统数据段(system segment)(包含PCB)组成的一个动态实体

可运行队列



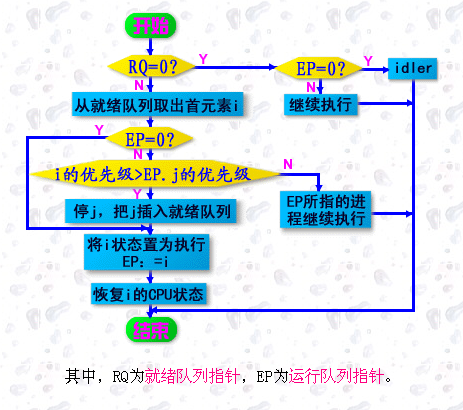


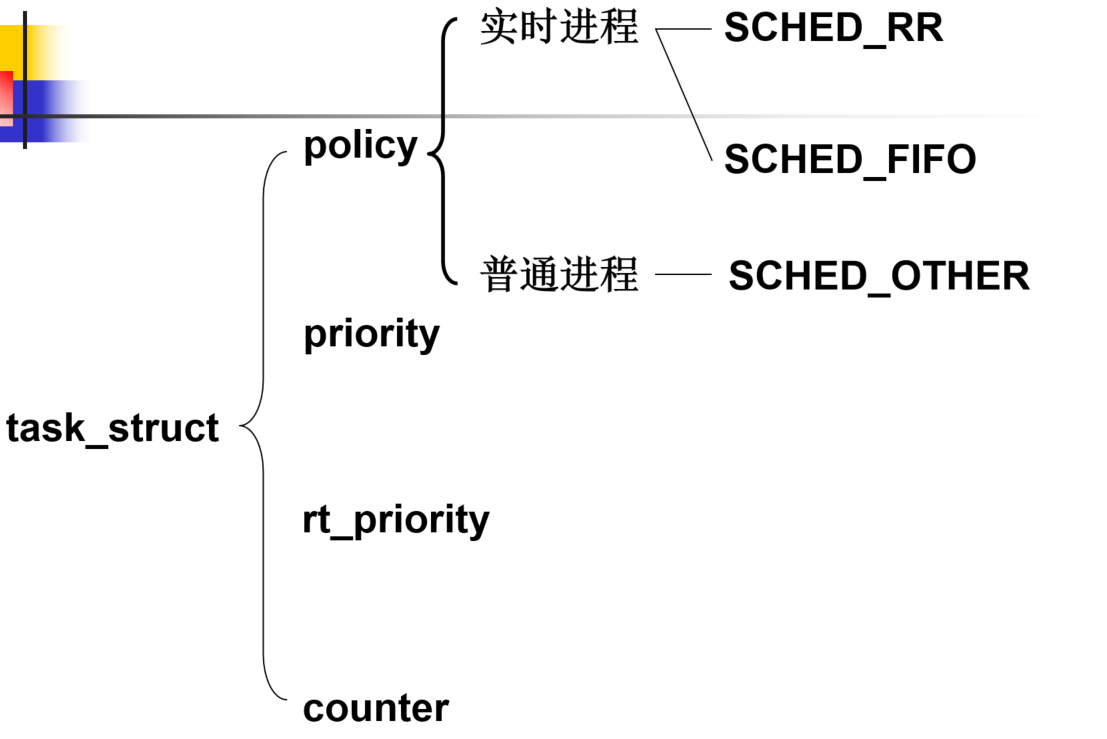


进程调度：非剥夺方式  剥夺方式  优先权原则、短进程、优先原则、时间片原则

最高优先权(FPF)优先调度算法：  
静态优先权    静态优先权是在创建进程时确定的，在整个运行期间不再改变动态优先权

动态优先权 是基于某种原则，使进程的优先权随时间改变而改变。





Counter 子为父的一半

rt\_priority

实时进程的优先级，可通过系统调用改变

priority

进程静态优先级，给出进程每次获取cpu后可使用的时间（按jitty计算）

Counter

表示进程当前还可运行多久

进程开始运行时被赋为priority值，以后，每隔一个tick（时钟中断）递减1，减到0时引起新一轮调度。

重新调度将从runqueue队列中选出运行运行权值最大的就绪进程获得cpu。

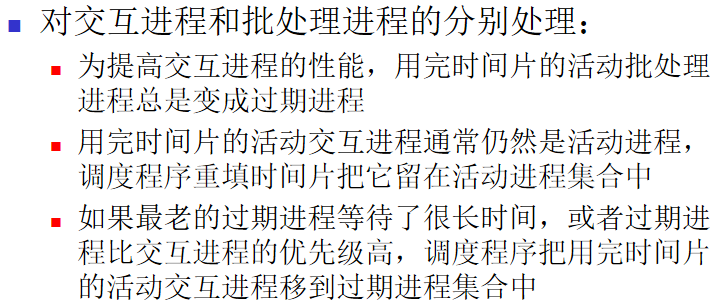
创建一个新的进程时，子进程会继承父进程的一半剩余时间片

Linux的进程调度是基于优先级的调度

Linux的进程分为普通进程和实时进程，在基于优先级的算法下实时进程的优先级高于普通进程。

Linux中进程的优先级是动态的，调度程序周期性的调整他们的优先级，避免进程饥饿

|  |  |  |
| --- | --- | --- |
| 符号常量 | 意义 |  |
| SCHED\_OTHER | 普通进程的时间片轮转算法 | 时间片用完，放在过期队列中，提高优先权，时间戳， |
| SCHED\_FIFO | 实时进程的先进先出算法 | 一旦占用cpu则一直运行。一直运行直到有更高优先级任务到达或自己放弃 |
| SCHED\_RR | 实时进程的时间片轮转算法 | 进程的时间片用完，系统将重新分配时间片，并置于就绪队列尾 |



实时进程运行后会一直占用CPU资源，只有下列情况在会被另一进程替代：

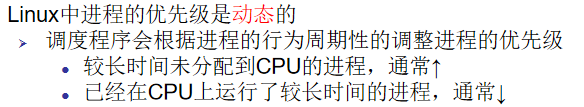
进程被更高优先级的实时进程取代

进程执行了阻塞操作并进入睡眠

进程停止或被杀死

进程自愿放弃cpu（可以通过调用系统调用sched\_yield()放弃CPU）

进程是基于时间片轮转的实时进程，且用完了时间片



1. **定时器中断，中断**

定时器溢出而申请的中断

解决外设与cpu速度不匹配的问题。

当一个进程在执行时,CPU的所有寄存器中的值、进程的状态以及堆栈中的内容被称为该进程的上下文。当内核需要切换到另一个进程时，它需要保存当前进程的所有状态，即保存当前进程的上下文，以便在再次执行该进程时，能够必得到切换时的状态执行下去。

在LINUX中，当前进程上下文均保存在进程的任务数据结构中。在发生中断时,内核就在被中断进程的上下文中，在内核态下执行中断服务例程。但同时会保留所有需要用到的资源，以便中继服务结束时能恢复被中断进程的执行。

普通进程可以被中断或异常处理程序打断

异常处理程序可以被中断程序打断

中断程序只可能被其他的中断程序打断

中断：

1) 将下一条指令的地址存入相应连接寄存器LR，以便程序在处理异常返回时能从正确的位置重新开始执行。

2) 将CPSR复制到相应的SPSR中。

3) 根据异常类型，强制设置CPSR的运行模式位。

4) 强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

恢复：

1) 将连接寄存器LR的值减去相应的偏移量后送到PC中。

2) 将SPSR复制回CPSR中。

3) 若在进入异常处理时设置了中断禁止位，要在此清除

流程

1，若有必要，进入核心态

2，在内核态堆栈保存上下文（用户态／核心态）

3，调用asm\_do\_IRQ #处理中断

4，恢复上下文

5，若有必要，返回用户态

1. **设备驱动如何与软件挂钩**
2. **FD，文件控制块**
3. **Open系统调用的具体过程**
4. **u-boot板移植，内核移植挂载**
5. **内核如何管理驱动程序**
6. **编程题，（只讲过一个编程？）**
7. **下载到开发板不能运行**

缺库 工具查找相应的文件依赖，拷贝依赖库