南 京 理 工 大 学

软件课程设计三

|  |  |  |  |
| --- | --- | --- | --- |
| **姓 名:** | 蒋旭钊 | **学 号:** | 918106840727 |
| **学院(系):** | 计算机科学与工程学院 | | |
| **专 业:** | 计算机科学与技术 | | |
| **课 程:** | 软件课程设计III | | |

2021 年 12 月

**1. 内核模块**

在内核模块上，我在JXZInnerModule目录下编写了Process.c文件，实现了打印当前系统的所有进程的程序，并最终编译成Process.ko模块注入了系统中，最后将固件打包配置到带有Padavan的YK-L1路由器上验证成功。

这里有一个技巧，我在/root/test/padavan-ng/trunk/linux-3.4.x/drivers目录下建立自己的JXZInnerModule目录。打开trunk目录下的build\_firmware.sh的源码我们可以看到，这里主要根据.config里面已有的各种CONFIG配置进行和YK-L1硬件挂钩的二进制流烧制，最后可以注意到有make命令，这是根据trunk文件夹里面的Makefile文件相关的，主要进行二进制流烧制前各类modules的编译工作，其中比较重要的代码有：

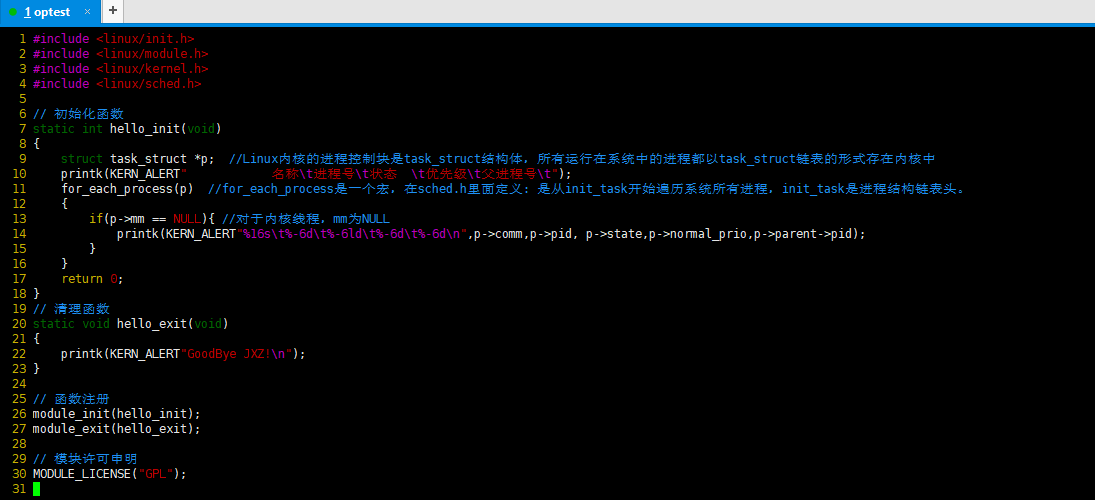
**ARCH\_CONFIG = $(ROOTDIR)/vendors/config/mips/config.arch**

它在config.arch里面指定了最终编译的文件是和MIPS系统有关的。

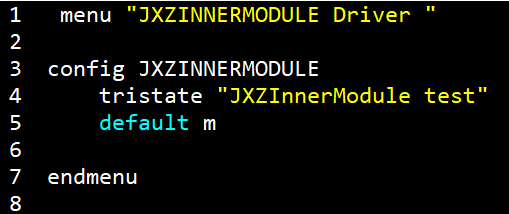
**.PHONY: modules**

它定义了Makefile要编译的各类模块，其中会根据各个模块中的.config文件进行配置，.config文件又和各个文件夹中的Kconfig文件有关，上面的drivers驱动模块目录下也包含着这一系列文件。而Kconfig文件可以让我们在linux-3.4.x目录下使用menuconfig指令调出模块选择的操作，最终可以实现模块的移植。

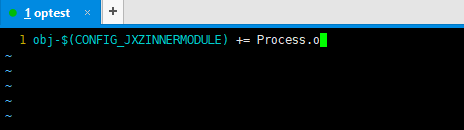
所以最终我们只需要在JXZInnerModule下编写Process.c的Makefile和Kconfig文件，然后在drivers目录下修改Makefile和Kconfig文件加上我们刚刚的JXZInnerModule目录和其下的Kconfig文件，就能最终打包编译成集成了我们的模块的固件。



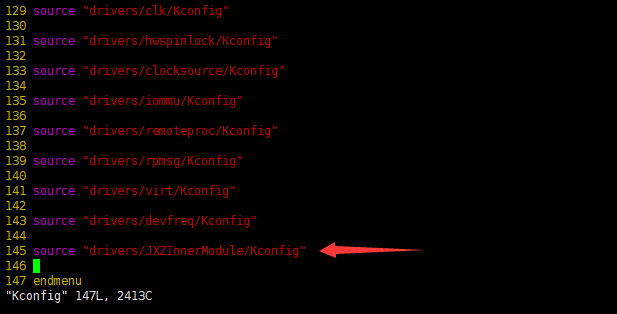
**图1. Process.c文件**



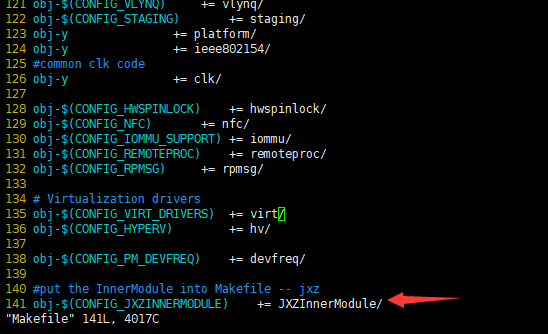
**图2. JXZInnerModule下的Kconfig文件**



**图3. JXZInnerModule下的Makefile文件**

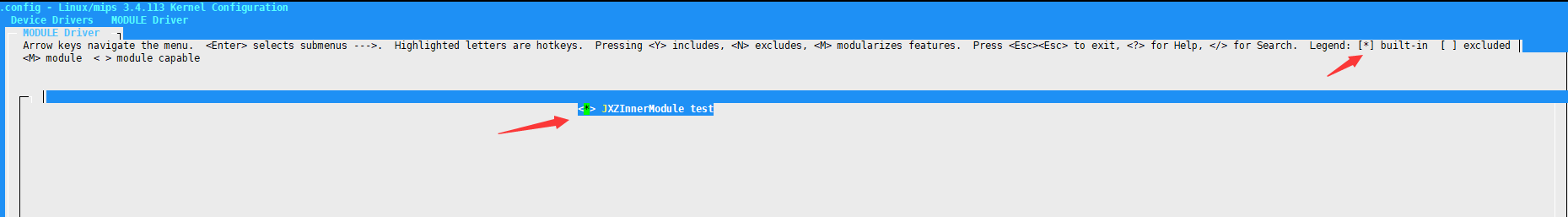


**图4. drivers下的Kconfig文件**



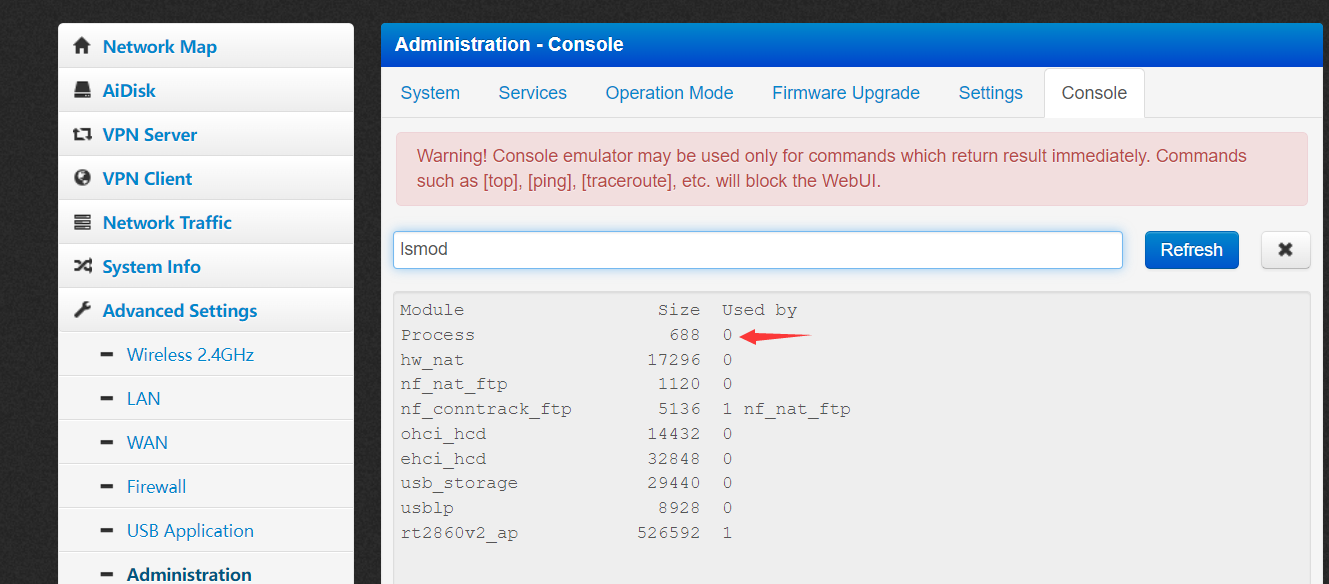
**图5. drivers下的Makefile文件**

通过在linux-3.4.x下执行make menuconfig命令，选取我们的JXZInnerModule test模块：

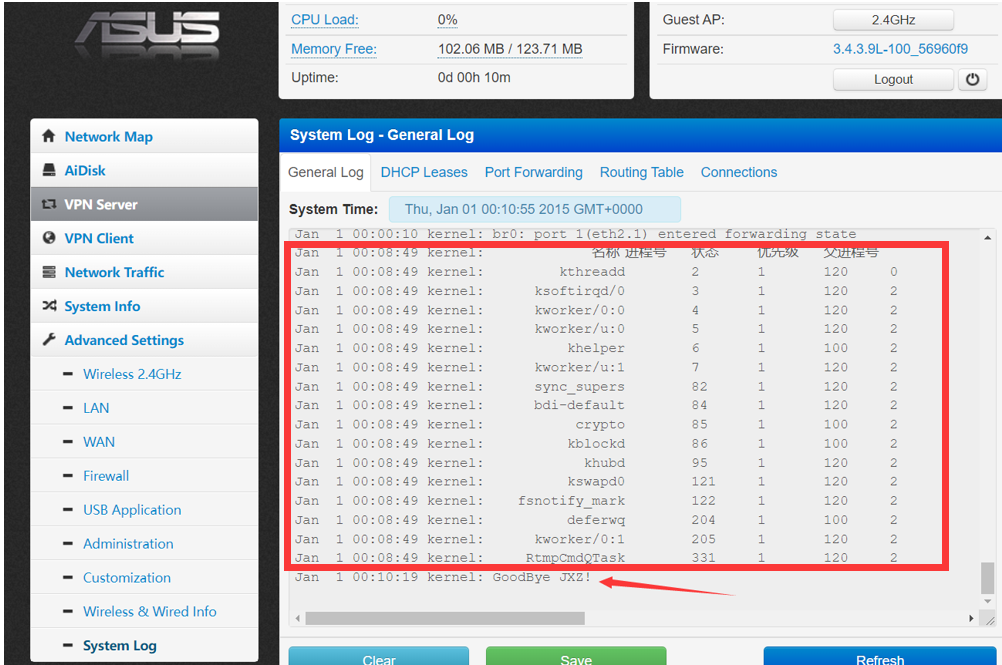


**图6. Linux-3.4.x下的menuconfig界面**

最终我们用路由器进行烧制，得到的结果如下，这里有一个区别，如果我们在编写Process.ko的Kconfig文件时，如果我们代码写“default m”代表最终模块是需要我们自己插入的。在Administration的Console下面我们通过find命令找到Process.ko模块，然后insmod进系统，通过lsmod发现已经插入以后，就可以在System Log中看到所有进程信息。最后用rmmod命令可以卸载模块，也可以看到打印信息。

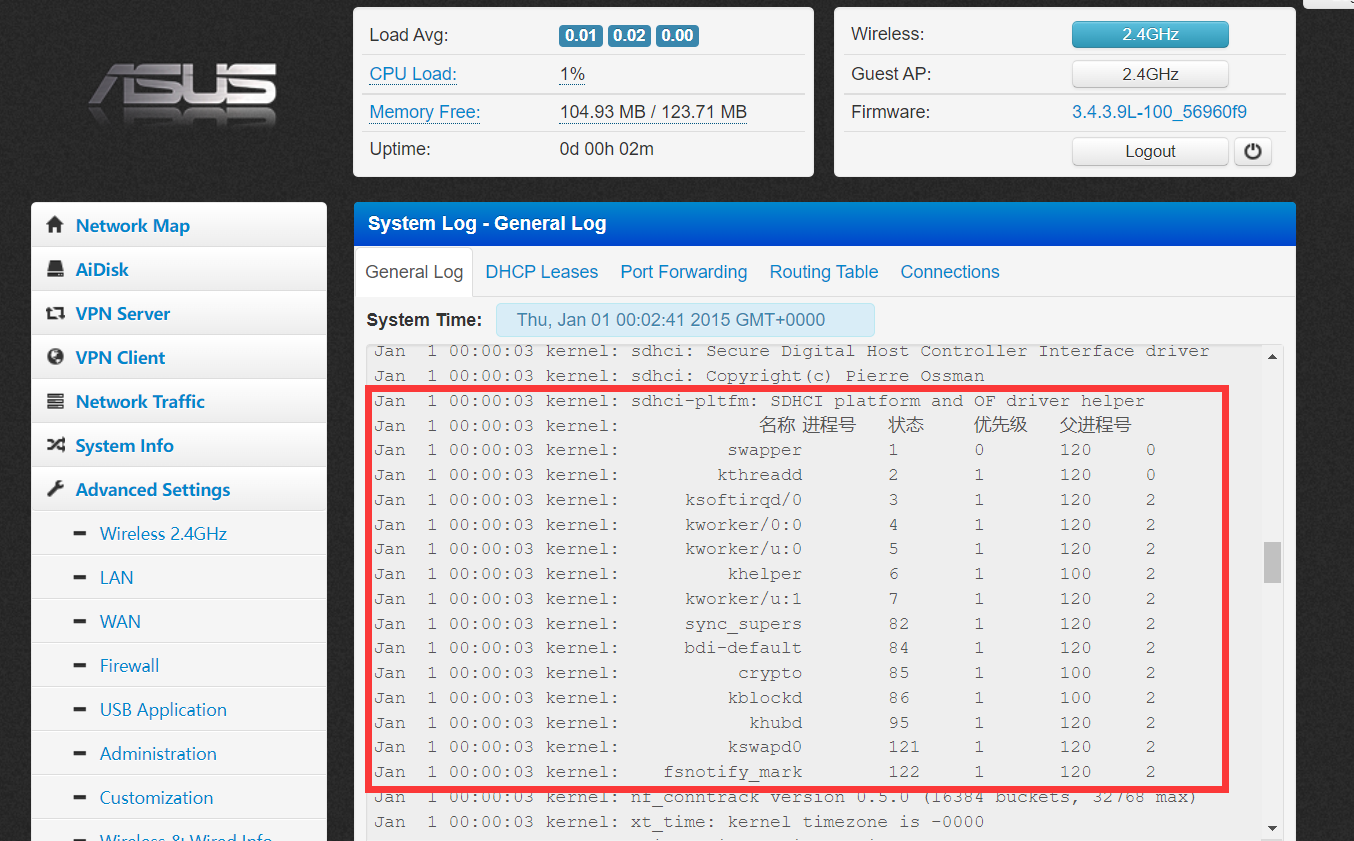


**图7. Padavan下lsmod(default m)**



**图8. Padavan下的System Log(default m)**

如果我们在Kconfig中“default y”则表示模块已经built-in了，可以随系统一起启动。



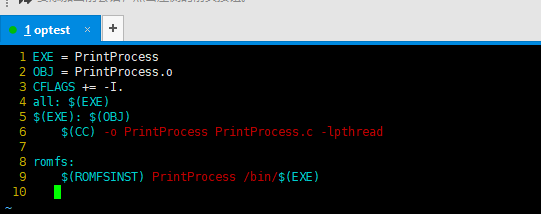
**图9. Padavan下的System Log(default y)**

**2. 应用模块**

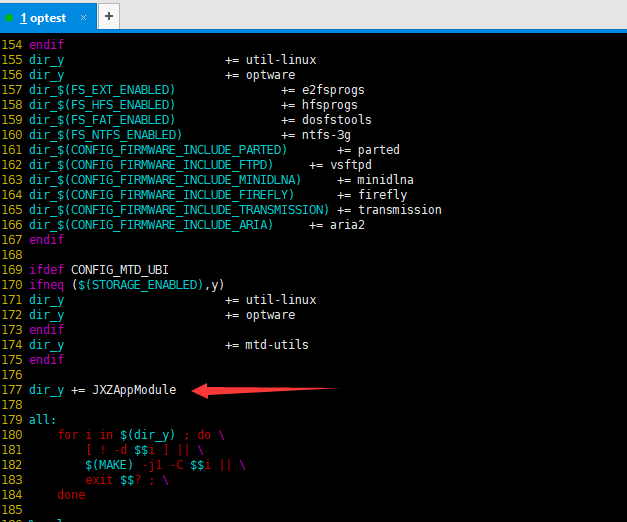
应用模块中我先在/root/test/padavan-ng/trunk/user下新建了JXZAppModule文件夹，然后编写了PrintProcess.c文件，在其中通过pthread\_create创造了两个线程并打印输出信息，最后编译成PrintProcess.exe。在user目录下修改Makefile文件包含此目录，最后即可通过Build\_firmware.sh将应用程序打包。



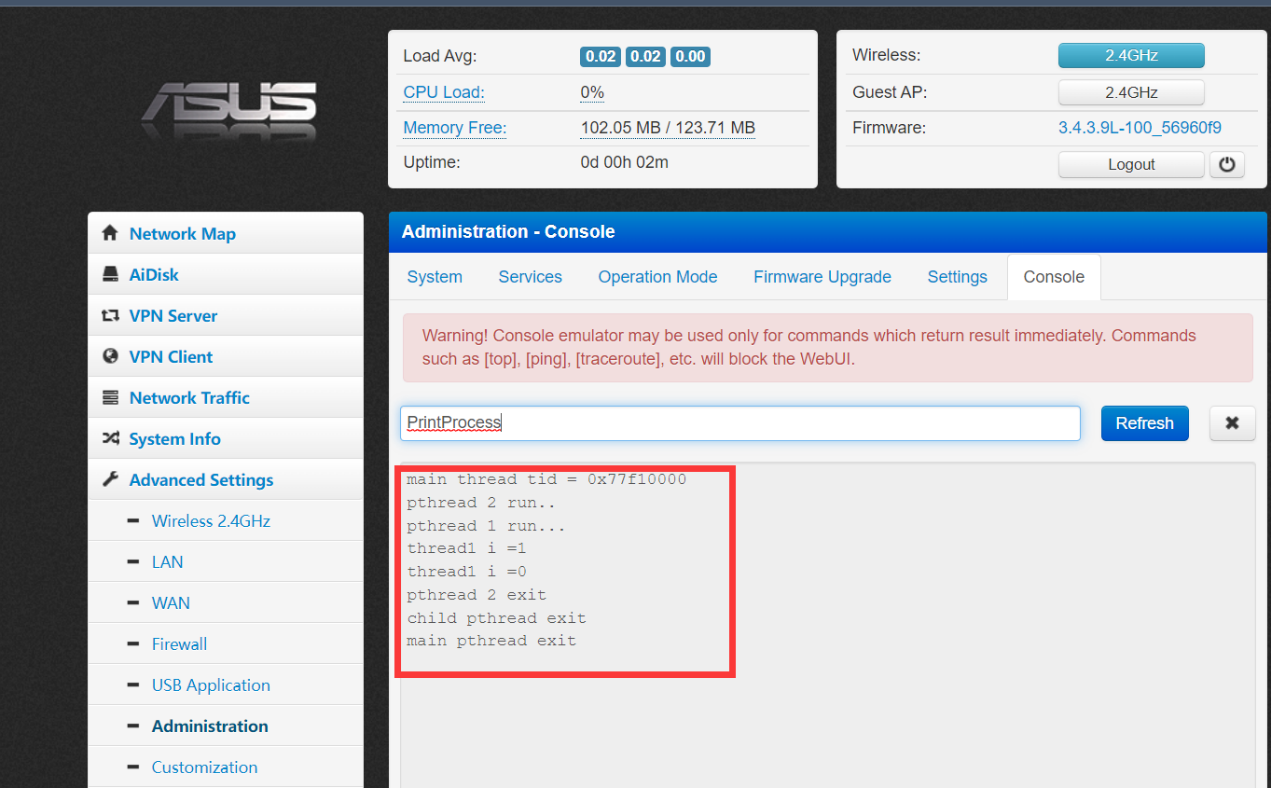
**图10. PrintProcess.c文件**



**图11. JXZAppModule下的Makefile**



**图12. user下的Makefile**



**图13. 应用模块的运行结果**

**3. 理解与流程图**

Core.c文件中主要涉及了进程调度的相关知识，但是Linux的编程代码与迭代博大精深，想在短时间内完全琢磨透是很难的，因此需要我们去把握重点。如果要把握重点的话，需要从其中使用的主要数据结构以及主要调度流程说起，希望最后能够梳理一个清晰的脉络出来。

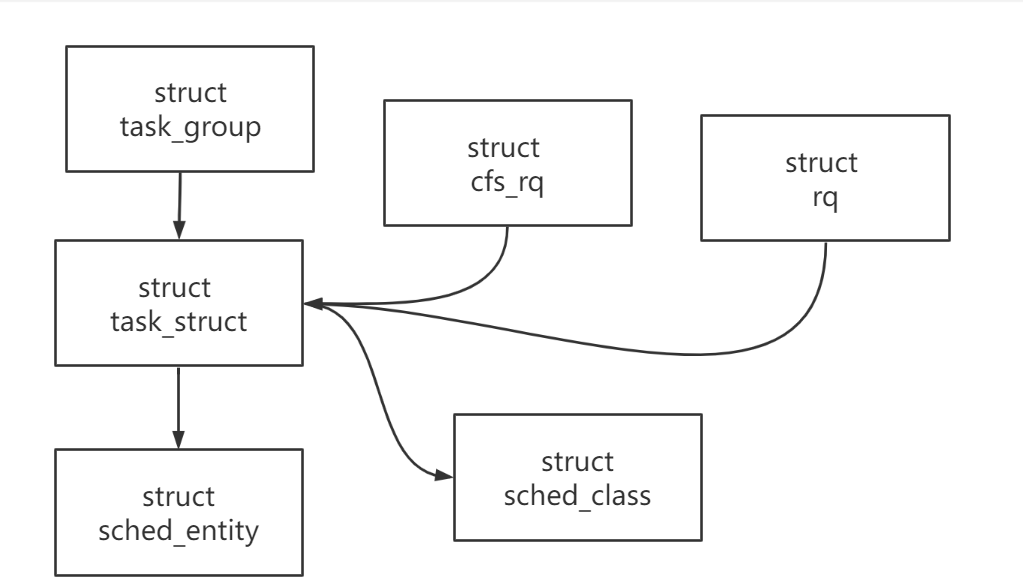
**核心数据结构：**

* 每个CPU对应包含一个运行队列结构(struct rq)，而每个运行队列又包含有其自己的实时进程运行队列(struct rt\_rq)、普通进程运行队列(struct cfs\_rq)、和dl队列(struct dl\_rq)。
* 组调度(struct task\_group)：在多核多CPU的情况下，同一进程组的进程有可能在不同CPU上同时运行，所以每个进程组都必须对每个CPU分配它的调度实体(struct sched\_entity 和 struct sched\_rt\_entity)和运行队列(struct cfs\_rq 和 struct rt\_rq)。
* 调度实体(struct sched\_entity)：其中，load：权重，通过优先级转换而成，是vruntime计算的关键。vruntime：虚拟运行时间，调度的关键，其计算公式：一次调度间隔的虚拟运行时间 = 实际运行时间 \* (NICE\_0\_LOAD / 权重)。vruntime最小，最小的将被调度。cfs\_rq：此调度实体所处于的CFS运行队列。调度实体(struct sched\_entity)，其被包含在 struct task\_struct 结构中的se中。
* struct task\_struct： 进程优先级，int prio, static\_prio, normal\_prio,实时进程优先级,unsigned int rt\_priority;调度类，调度处理函数类,const struct sched\_class \*sched\_class; 调度实体(红黑树的一个结点),struct sched\_entity se;调度实体(实时调度使用),struct sched\_rt\_entity rt;
* struct sched\_class（调度类）:调度类优先级顺序: stop\_sched\_class

 -> dl\_sched\_class -> rt\_sched\_class -> fair\_sched\_class ->

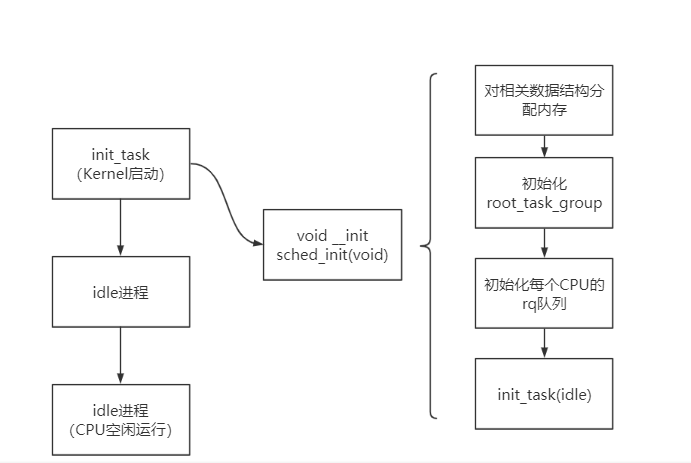
idle\_sched\_class,里面包含的是调度处理函数，enqueue\_task，将进程加入到运行队列中；dequeue\_task，从运行队列中删除进程。

* CFS运行队列(struct cfs\_rq)：只要确定其代表着一个CFS运行队列，并且包含有一个红黑树进行选择调度进程即可。
* CPU运行队列(struct rq)：其用于描述在此CPU上所运行的所有进程，其包括一个实时进程队列和一个根CFS运行队列，在调度时，调度器首先会先去实时进程队列找是否有实时进程需要运行，如果没有才会去CFS运行队列找是否有进行需要运行。



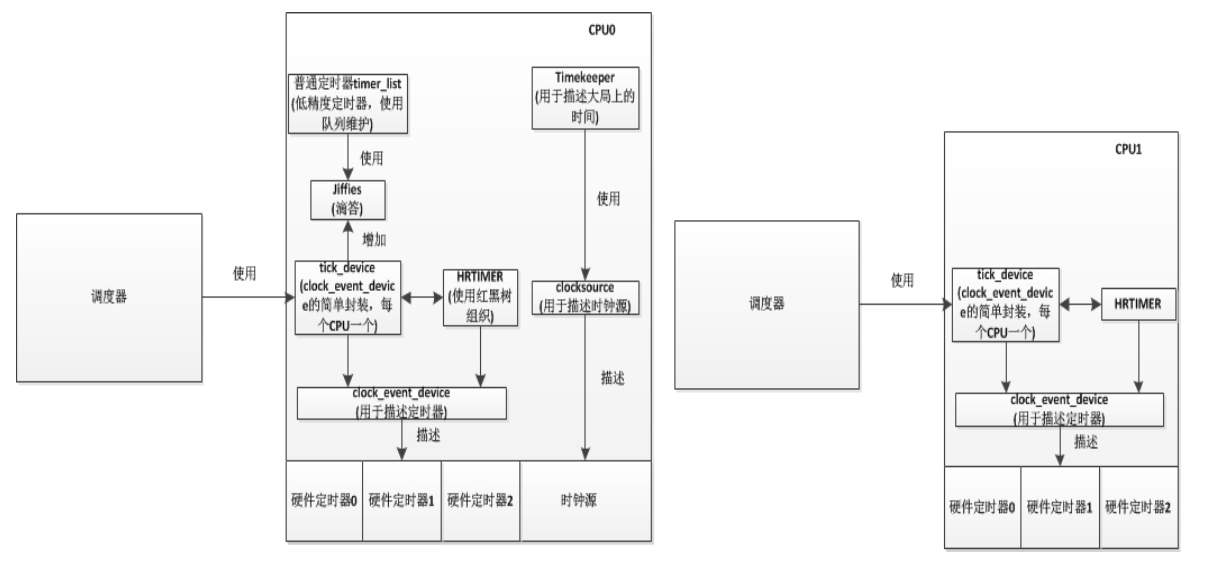
**调度器启动与初始化：**

在start\_kerenl函数中，进行了系统启动过程中几乎所有重要的初始化(有一部分在boot中初始化，有一部分在start\_kernel之前的汇编代码进行初始化)，包括内存、页表、必要数据结构、信号、调度器、硬件设备等，这些初始化是由init\_task这个进程负责的。　在start\_kernel中对调度器进行初始化的函数就是void \_\_init sched\_init(void)，其主要工作为：对相关数据结构分配内存alloc\_size，初始化root\_task\_group，初始化每个CPU的rq队列(包括其中的cfs队列和实时进程队列)，将init\_task进程转变为idle进程。



**调度器运行：**

调度器会涉及到内核中定时器的实现，因此需要了解如何通过定时器实现了调度器的间隔调度的。在系统中，每一次时钟滴答都会使调度器判断一次是否需要进行调度。



Linux 使用的是带时间片的动态优先级抢占式调度模式, 被称之为公平调度 CFS 的算法。利用 nice 值+实时优先级+时间片共同维护线程的优先级，而这个优先级队列，也就是就绪态队列，Linux 是用红黑树来维护的。

