一：顶层makefile

1.确定编译模式，简单编译，静默编译，详细编译（根据命令行指令，看v的值）

2.使用“make xxx\_defconfig”与%config相匹配，配置 Linux 内核

二：linxu内核启动流程

**内核启动被分成了两个阶段，第一阶段先执行 initrd 文件系统中的"某个文件"，完成加载驱动模块等任务，第二阶段才会执行真正的根文件系统中的 /sbin/init 进程**  
  A.第一阶段(head.s,汇编语言，传递启动参数)

##### ****汇编函数中主要操作****

* ****使能SVG模式(处理器的特权模式,例如uboot启动的svc32模式)****
* ****关闭所有中断(确保在初始化阶段没有中断请求干扰启动流程)****
* ****校验 atags 或设备树(dtb)的合法性(帮助内核识别和初始化设备)****
* ****使能MMU(内存单元:虚拟地址到物理地址的转化,实现内存的保护和分页)****

#### B.第二阶段(init/main.c)

1. Init():系统的各种初始化，比如内存，定时器，系统时间等

在stext末尾调用rest\_init()，这个函数创建了3个进程，分别是idle，kernel\_init，kthreadd。记作0-2号进程

1号:**kernel\_init主要任务是完成设备驱动初始化和挂载根文件系统，并读取根文件系统中init程序，将从内核态转变到用户态**

**2号:内核线程的守护进程，始终运行在内核态，负责所有内核线程的创建。原理是不断循环kthread\_create\_list全局链表，如果为空，则调度出去；否则则调用create\_kthread接口来创建内核线程。**

**3号:系统运行时，保持系统的空闲状态和节省资源**

1. Linux移植：  
    A.修改顶层makefile:  
    在顶层的makefile直接定义 ARCH(目标框架) 和CROSS\_COMPILE(交叉编译) 为arm 和arm-linux-guneabihf-  
    B.配置并编译linux内核:
2. 先清理上一次内核 make clean
3. 配置内核(找到对应开发板的默认配置文件)
4. 这时make -j16开始编译，完成后会得到zImage镜像文件和开发板对应的dtb  
   (由.dts设备树文件编译出来的二进制文件)
5. 内核启动测试(驱动手册P976,包含启动命令等,很可能遇到根文件缺少错误)
6. 在linux添加开发板

1.添加开发板默认配置文件，将arch/arm/configs里面的xxx\_defconfig复制后重 命名，修改重命名后的文件内容(修改内容参考驱动手册P978) ，完成后，即 可make配置Linux内核

2.添加开发板对应的dts，复制合适的dts后重命名，然后arch/arm/boot/dts/makefile文件中，找到“$(CONFIG\_SOC\_IMX6ULL)”,将重命名后的文件加入

3.开始编译测试(驱动手册P980)

D.可能涉及到开发板的CPU驱动和网络驱动的修改(详细看驱动手册P981)

E.最后一步,根文件系统构建(详细参考根文件系统相关知识,单独一个文档)