嵌入式 Linux 系统与单片机比较及快速搭建系统

作为入门，首选 JZ2440 开发板，资料丰富，视频齐全。
原理是相同的，使用 JZ2440 所学知识完全适用于其他板子。
本文使用 nanoPI 作为例子，所涉及主要知识都来自 JZ2440 的资料及视频。

网站/论坛：www.100ask.net，www.100ask.org
淘　　宝：100ask.taobao.com
邮　　箱：weidongshan@qq.com
微信公众号：baiwenkeji
公司　微博：百问科技
个人　微博：韦东山

本文档及视频涉及的所有资料，请根据 www.100ask.net 下载页面的说明，从网盘上下载。

类别	内容
关键词	单片机、嵌入式 linux
摘要	1）单片机与 linux 系统的对比 2）嵌入式 Linux 系统组成 3）怎么快速构建自己的根文件系统，以 nanoPI 为例
当前版本	V1
编辑	韦东山，王辉，黄成
审核	韦东山

修订历史

版本	更改日期	更改说明
v1	2017.04.18	初建，2017.04.19 腾讯直播前夜发布

1. 嵌入式 Linux 与单片机的比较

现在的电子专业，仍以单片机 C51 为基础教程，条件好点的使用 STM32 单片机。
单片机还有用吗？
有用！看看现在热卖的小米智能插座，它就是使用 Cortex-M4 核的 MW300 做的。
单片机容易学吗？
相对来说，单片机挺容易学的。

问题来了，容易学的东西，会的人就多；
会的人多了，工资就低。
为了前途，为了钱途，我们必须升级！

从知识的角度来说，会 Linux 操作系统的人，肯定会单片机；反过来，会单片机的人，不一定会 Linux。

所以，如果你对电子专业有兴趣，开始学习嵌入式 Linux 吧。

那还要不要学习单片机？对于这个问题，我有专门的论述：http://100ask.org/a/howtostudy/，请看“2.2.4 要不要专门学习 Windows 下的单片机开发”。

1.1 以全志 H3、STM32F407 为例进行比较

以下是芯片资源的比较：

	STM32F405xx-407xx	全志H3
CPU	Cortex-M4, 最高频率168MHz FPU	Cortex-A7, 4核, 1.2GHz FPU: VFPv4
内存系统	板载：1M Flash, 192+4K SRAM	板载：96KROM，不可编程
对外存储接口	SRAM, PSRAM, NAND, NOR，SD	NAND, SD/TF卡, eMMc, NOR DDR2/DDR3/DDR3L/LPDDR2/LPDDR3, 多达2G
GPU	无	ARM Mali400MP2 GPU
Video Engine	无	H.265/HEVC 4K@30fps视频硬解 H.264 1080P@60fps
显示系统	并行接口, 8080/6800 modes	HDMI1.4，CVBS
安全加密系统	96-bit unique ID	DRM, 信息加密/解密, secure boot, secure JTAG, efuse
对外接口：		
A/D转换	3x12-bit, 2.4 MSPS	无
D/A转换	2x12-bit	无
TWI	无	1个
CIR	无	1个
定时器	17个，都带有IC/OC/PWM引脚	2个普通定时器，1个高速定时器，1个看门狗定时器
IO口	多达140个，都有中断功能	多达106个
I2S	无	2个
OWA(One Wire Audio)	无	1个
I2C	多达3个	
USART/UART	多达4个	多达5个
SPI	多达3个	多达2个
CAN	多达2个	无
USB 2.0	2个, host/device/otg	1个OTG控制器，3个USB HOST控制器
网口	10/100M Ethernet MAC	10/100/1000Mbps EMAC
Camera	8-to 14-bit parallel	parallel CMOS sensor interface
RTC	1个	1个
SCR(Smart Card Reader)	无	1个

最直观的区别：主频、内存、GPU、显示系统。

以下是开发板价格的比较：

STM32：

nanoPi:

可以得出这样的结论：

① 在硬件性能方面，H3 秒杀 STM32F407

② 在硬件价格方面，以核心板为例，它们差不多

③ 在对外接口方面，各有千秋

④ 在软件可扩展方面，H3 超越 STM32F407 太多

看了这些比较，你还会因为“精通 STM32”而自得吗？我见过很多工作多年的单片机工程师，30 多岁、40 多岁发邮件给我们团队、电话联系我们团队，希望从单片机转型到 linux 提供一些学习的建议和帮助。

如果你不具备特殊行业（如：医疗、工业）的相关算法（如：传感器、电机控制），你只会使用 STM32 点亮一个 LED 灯，会操作 UART、SPI、I2C 等外设，很可能下一个转型的就是你！

但是，请你千万不要误会，我没有看不起“单片机”。

设计产品时，我会选择最优方案：

① 如果使用 1、2 元钱的 C51 可以实现同样的功能，我干嘛选用 30 元的 STM32，干嘛选用更贵的嵌入式 Linux？

② 性价比相同之下，我当然选择开发难度更小的单片机

③ 并且对于功耗要求更低的系统，经常是“单片机+嵌入式 Linux”组合

我只是从学习的角度说，你掌握了嵌入式 Linux，再回头很容易掌握单片机；而掌握了单片机，需要再升级一下进入 Linux 的世界。

1.2 实际产品对比嵌入式 Linux、单片机

	单片机 如 51、STM32、AVR、MSP430 等	嵌入式 linux SOC 芯片 如 s3c2440、全志 H3、s3c4412
主频	51 ：12Mhz STM32F1：72Mhz STM32F4：128Mhz	S3c2440:单核 400Mhz 全志 H3:4 核 1.2G s3c4412: 4 核 1.4G
内存	内部自带 RAM，一般 32KB—128KB	SDRAM、DDR2、DDR3 常见：64MB、256MB、512MB
存储设备	Flash，一般为 64KB ---512KB	Nand flash、emmc 常见 256MB、512MB、1GB
显示设备	一般为 TFT 显示屏 缺点：由于主频限制，在切换图片显示时，卡顿不流畅。	可外接 VGA 显示器、HDMI 显示器、消费类电子产品，越来越趋向于分辨率大，多屏显示
摄像头	有些高端一点的单片机可外接 cmos 摄像头 缺点：由于主频限制，对摄像头的数据处理仍然卡顿	Usb 接口的摄像头、mipi 接口的摄像头
网络	使用简单的网络协议栈如 lwip 缺点：网络应用程序太少，没有实现完整的网络协议栈	完整的网络协议栈、 应用程序：socket 编程，HTTP 服务器、PHP 服务器

所以：
单片机只能适合于简单的命令控制、数据采集、数据处理、数据传输等场合。
嵌入式 Linux 适用于人机交互的场合，适用于多任务的场合。

以下是一些产品。


常见Linux、安卓应用1
路由器
行车记录仪
考勤机
NOC INTERCONNECT
KINDLE电子书
电子相册
监控摄像头


常见Linux、安卓应用2
电视盒子
MEMORY
CPU CORES
手持终端PDA
双屏收银机
智能手机
广告机


常见Linux+单片机应用
充电桩
心电图监护仪
扫地机器人
无人机

2. 嵌入式 Linux 系统上的第 1 个程序

2.1 编写、编译 HelloWorld

hello.c:

```
#include <stdio.h>
int main(int argc, char **argv)
{
    printf("Hello, world!\n");
    return 0;
}
```

编译：

```
$ gcc -o hello hello.c    // 为 PC 编译
$ ./hello                 // 测试一下
$ arm-linux-gcc -o hello hello.c   // 为 ARM 编译，这个程序需要放到开发板上
```

注意：要使用 arm-linux-gcc 编译，需要先安装交叉编译工具链，请参考后面的 4.1 节。

2.2 怎么执行：放到板子上，然后运行

可以通过 U 盘，也可以通过 NFS。

开发过程中，多用 NFS，调试很方便

2.3 想自动运行，怎么办？

你要先看完第 4 课，制作好自己的根文件系统后，再修改/etc/inittab 或/etc/init.d/rcS。

比如对于 hello 程序，可以这样修改：

(1) hello 程序只执行一次，修改/etc/inittab，添加这行：

```
::once:/bin/hello > /tmp/hello.log  2>&1
```

它的含义是只执行 hello 程序一次，里面的 printf 信息写到/tmp/hello.log(文件句柄为 1)；“2>&1”表示如果有错误信息(文件句柄 2)，也打印到文件句柄 1 对应的文件/tmp/hello.log 去。

(2) hello 程序退出后又重新执行，修改/etc/inittab，添加这行：

```
::respawn:/bin/hello > /tmp/hello.log  2>&1
```

(3) hello 程序只执行一次，修改/etc/init.d/rcS，添加这行：

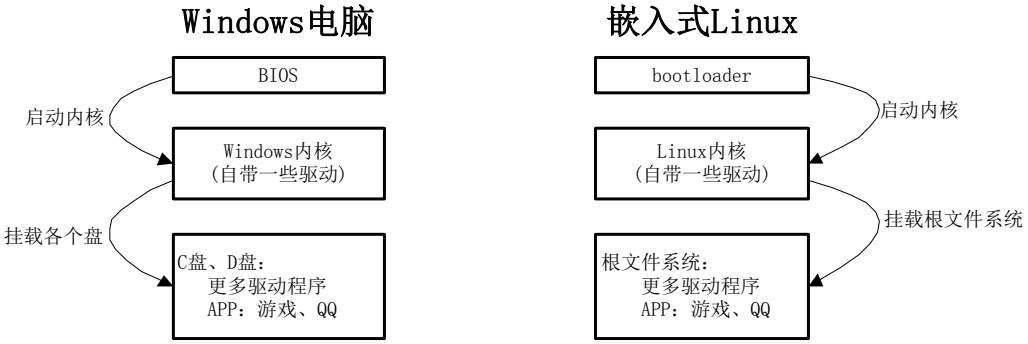
```
/bin/hello > /tmp/hello2.log 2>&1  &
```

最后的“&”表示让 hello 程序在后台运行。

3. 嵌入式 Linux 系统组成

3.1 系统组成

嵌入式 Linux 系统包含哪些东西？不要急，举一个例子你就知道了。



① PC 里，BIOS 首先运行，做一些自检，然后启动 Windows；
嵌入式 Linux 里，对应的程序被称为 bootloader，它的目的也是启动 Linux 内核

② 操作系统内核：Windows、Linux
提供文件管理、内存管理、进程调度、驱动程序等功能；
工作中，我们一般会编写驱动程序，通过驱动程序访问硬件。

③ 我们买电脑、手机、平板，目的是为了使用各种 APP：游戏、QQ、微信
这些程序在哪里？
对于 PC，它位于 C 盘、D 盘上；C 盘是系统盘，存放各种系统文件；D 盘给我们自己用
对于嵌入式 Linux，这些系统文件位于“根文件系统”里，挂载了“根文件系统”后，可以再挂载位于其他分区上的其他“文件系统”。

所以，嵌入式 Linux+Android 系统包含以下 3 部分内容：

(1) bootloader
(2) Linux 内核
(3) 根文件系统：里面含有系统文件、驱动程序、应用程序

3.2 分析 nanoPi 的刷机包

分析依据：友善之臂官方 WIKI
http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO/zh
http://wiki.friendlyarm.com/wiki/index.php/NanoPi_M1/zh

(1) 更新 u-boot:

WIKI 中说更新 TF 卡上的 U-boot 的方法为：

```
$ ./fuse_uboot.sh /dev/sdx
```

分析 fuse_uboot.sh，可知它实际上做如下事情：

```
boot0_fex=boot0_sdcard.fex
uboot_fex=u-boot.fex

cd tools/pack/out/ > /dev/null
[ -e ${boot0_fex} ] && dd if=${boot0_fex} of=${SDCARD} bs=1k seek=8
[ -e ${uboot_fex} ] && dd if=${uboot_fex} of=${SDCARD} bs=1k seek=16400
sync
```

结论：

A. bootloader 分为两部分：boot0_sdcard.fex、u-boot.fex
B. 分别位于 SD 卡上偏移地址 8K、16400K 处，大小分别有 32K、900K 左右。

(2) 更新内核：

WIKI 中说更新 TF 卡上的内核的方法为，先编译：

```
$ ./build.sh -p sun8iw7p1 -b nanopi-h3 -m kernel
```

编译完成后内核 boot.img 和驱动模块均位于 linux-3.4/output 目录下，将 boot.img 拷贝到 TF 卡的 boot 分区的根目录即可。

分析整个过程，发现它编译内核的方法为：

```
$ cd linux-3.4
$ cp arch/arm/configs/sun8iw7p1smp_defconfig .config
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j4 uImage modules
```

上述命编译出了内核映象 uImage、bImage，也编译出了各个模块(.ko)文件；

最后，把 bImage、一个 RAMFS 根文件系统 rootfs.cpio.gz 一起打包为 boot.img。

更新内核时，把 boot.img 放入 SD 卡的第 1 个分区。

如果我们要重新配置内核，可以这样做：

```
$ export PATH=/your_lichee_dir/brandy/toolchain/gcc-arm/bin:$PATH
$ cd linux-3.4
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-  menuconfig
$ cd ..
$ ./build.sh -p sun8iw7p1 -b nanopi-h3 -m kernel
```

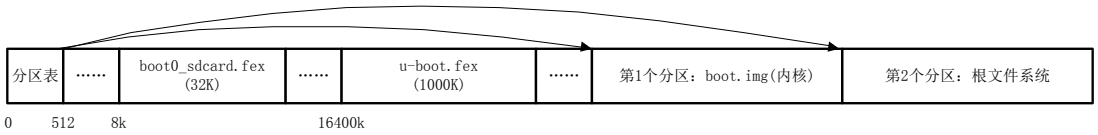
然后把 linux-3.4/output 目录下的 boot.img 复制到 SD 卡的第 1 个分区里。

(3) 更新文件系统

WIKI 上没有说明，但是可以从启动信息看出，它位于 SD 卡第 2 个分区，启动信息如下 ：

```
Kernel command line: console=ttyS0,115200 console=tty0 root=/dev/mmcblk0p2
rootfstype=ext4 rootwait init=/sbin/init
```

总结，nanoPI 的 SD 卡里，有如下内容：



4. 构造最小根文件系统

4.1 自己制作最小根文件系统

参考《嵌入式 Linux 应用开发完全手册》“第 17 章 构建 Linux 根文件系统”，韦东山 Linux 视频第 1 期的第 11 课。看完这一章书，一课视频，相信你就可以完全掌握根文件系统了。

下面只是罗列出制作根文件系统所用的命令。

(1) 准备材料：交叉编译工具链、busybox。

全志提供的编译器只能编译内核，无法编译应用程序，我们从友善之臂官网下载新的编译器。它的 WIKI 上是这样说明的：

```
$ git clone https://github.com/friendlyarm/prebuilts.git
$ sudo mkdir -p /opt/FriendlyARM/toolchain
$ sudo tar xf prebuilts/gcc-x64/arm-cortexa9-linux-gnueabihf-4.9.3.tar.xz -C
/opt/FriendlyARM/toolchain/
```

但是在国内访问 github 实在太慢，我把它上传到 git.coding.net 去了，可以用以下命令下载、安装：

```
$ git clone https://git.coding.net/weidongshan/nanoPI_app_toolchain.git
$ sudo mkdir -p /opt/FriendlyARM/toolchain
$ sudo tar xf  \
 nanoPI_app_toolchain/gcc-x64/arm-cortexa9-linux-gnueabihf-4.9.3.tar.xz \
  -C /opt/FriendlyARM/toolchain/
```

以后使用这个编译前，需要先设置 PATH 环境变量：

```
$ export PATH=/opt/FriendlyARM/toolchain/4.9.3/bin/:$PATH
```

busybox：从官网 https://busybox.net/，下载 busybox-1.26.2.tar.bz2

(2) 编译安装 busybox

看直播视频，下面是简单的笔记。

```
$ tar xjf busybox-1.26.2.tar.bz2
$ cd busybox-1.26.2/
$ make menuconfig // 设置交叉编译工具链前缀为 arm-linux-, 并选择支持 mount NFS
$ make
$ make install
```

(3) 从交叉编译工具链中复制 lib 库

```
$ cp -rf _install/ /work/nfs_root/fs_mini_nanoPI
$ mkdir /work/nfs_root/fs_mini_nanoPI/lib
$ cd /opt/FriendlyARM/toolchain/4.9.3/arm-cortexa9-linux-gnueabihf/lib
$ cp *so* -d  /work/nfs_root/fs_mini_nanoPI/lib
```

(4) 构建 etc 目录：

只需要创建 3 个文件：etc/inittab、etc/init.d/rcS、etc/fstab。

内容如下。

etc/inittab:

```
# /etc/inittab
::sysinit:/etc/init.d/rcS
ttyS0::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

etc/init.d/rcS：

```
#!/bin/sh
mount -a
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
```

还要改变它的属性，使它能够执行：

```
$ chmod +x etc/init.d/rcS
```

etc/fstab：

```
# device      mount-point    type    options        dump  fsck order
proc            /proc        proc    defaults         0     0
tmpfs           /tmp         tmpfs   defaults         0     0
sysfs           /sys         sysfs   defaults         0     0
tmpfs           /dev         tmpfs   defaults         0     0
```

(5) 构建 dev 目录：

```
$ cd /work/nfs_root/fs_mini_nanoPI
$ mkdir dev
$ cd dev
$ sudo mknod console c 5 1
$ sudo mknod null c 1 3
```

(6) 其他空目录，比如 proc、mnt、tmp、sys 等，如下创建：

```
$ cd /work/nfs_root/fs_mini_nanoPI
$ mkdir proc mnt tmp sys var
```

4.2 修改配置文件自动启动 APP

修改文件系统中 etc/inittab 或 etc/init.d/rcS 即可，参考第 2 课。

5. 自制刷机包

5.1 准备工作

按照友善之臂官方 WIKI：

http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO/zh

http://wiki.friendlyarm.com/wiki/index.php/NanoPi_M1/zh

先下载源码包得到 lichee 目录，再下载交叉编译工具链并放入 lichee/brandy/toochain/目录下。你得到的可能是文件 h3-lichee-20170119.7z，在 ubuntu 下需要先安装 7zip 工具才能解压，可以用以下命令：

```
$ sudo apt-get install p7zip
$ 7zr  x  -r  h3-lichee-20170119.7z
```

然后执行如下命令编译：

```
$ cd lichee
$ ./build.sh -p sun8iw7p1 -b nanopi-h3
```

接着，对于 nanoPI NEO，执行以下命令打包系统组件：

```
$ ./gen_script.sh -b nanopi-neo
```

对于 nanoPI M1，执行以下命令打包系统组件：

```
$ ./gen_script.sh -b nanopi-m1
```

本文适用于 nanoPI NEO 和 nanoPI M1，它们的操作命令只有上述这一处差别。

5.2 制作映象文件，并分区

制作一个 256M 的空白映象文件，你可以制作得更小或更大：

```
dd if=/dev/zero of=fs_nanoPI_256M.img  bs=1M count=256
```

把映象文件设置为“回环设备”：

```
sudo losetup /dev/loop0 fs_nanoPI_256M.img
```

划出 2 个分区，注意第 1 个分区起始地址要大于(16400K+1000K)，我们取它的偏移地址为 20M，大小为 32M；第 2 个分区只要大过 fs_mini_nanoPI 即可，我们取为 128M。

```
$ sudo fdisk /dev/loop0
分区信息如下：
    Device Boot      Start         End      Blocks   Id  System
    /dev/loop0p1            40960      106495       32768   83  Linux
    /dev/loop0p2           106496      368639      131072   83  Linux
```

识别分区，并格式化：

```
$ sudo partprobe /dev/loop0
$ sudo mkfs.vfat -I /dev/loop0p1
$ sudo mkfs.ext4 /dev/loop0p2
```

5.3 烧写 bootloader

```
$ cd tools/pack/out/
$ sudo dd if=boot0_sdcard.fex of=/dev/loop0 bs=1k seek=8
$ sudo dd if=u-boot.fex of=/dev/loop0 bs=1k seek=16400
```

5.4 烧写内核

```
$ cd linux-3.4/output/
$ sudo mount -t vfat /dev/loop0p1 /mnt
$ sudo cp boot.img /mnt
$ sudo umount /mnt
```

5.5 烧写文件系统

```
$ cd  cd /work/nfs_root/fs_mini_nanoPI/
$ sudo mount /dev/loop0p2 /mnt
$ sudo cp * -rfd /mnt
$ sudo umount /mnt
```

5.6 卸载回环设备，烧写 SD 卡

```
$ sudo losetup -d /dev/loop0
```

至此，得到映象文件 fs_nanoPI_256M.img，可以使用工具 win32diskimager 把它烧到 SD 卡中，此 SD 卡即可用来启动 nanoPI。

5.7 辅助命令

本节的命令是在开发板上运行的。

(1) 挂载 NFS：

```
# ifconfig eth0 192.168.1.18
# mount -t nfs -o nolock,vers=2 192.168.1.123:/work/nfs_root/fs_mini_nanoPI  /mnt
```