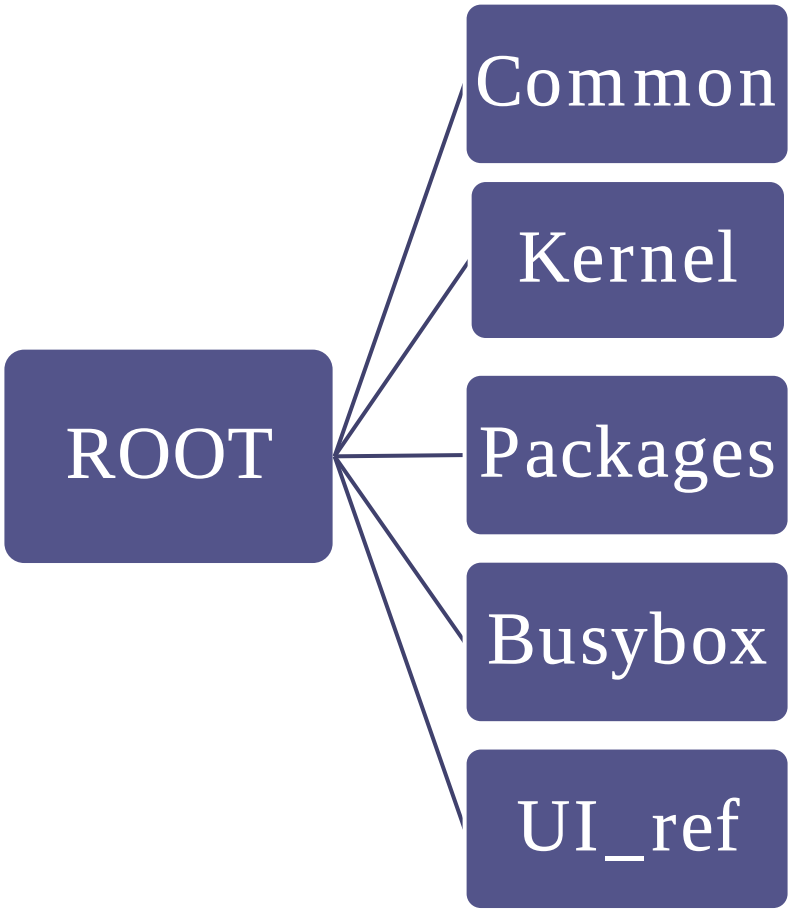
Linux 编译环境

Amlogic Beijing
Zhouzhi
2009-12-7

主题

- 基本目录结构和功能
- Kernel目录结构
- Common目录结构
- Packages 目录结构
- Ui_ref目录结构
- 主要编译命令
- 工具链

基本目录结构



基本目录结构

- [Kernel]:内核代码
- [Busybox]: Linux下简单基本的工具集
- [Packages]:应用程序软件包,包括基本的开源库,和我们测试代码,演示程序;
- [Common]:Makefile和编译工具以及不同开发板的配置代码;
- [ui_ref]:用户开发目录

Kernel 目录结构

- [Arch]架构相关代码,包括arm,mips,sh,spark,x86,
- [Block]块设备基础代码;
- [Crypto]加密相关
- [Documentation]kernel相关文档
- [Drivers]驱动相关代码
- [Fs]文件系统相关代码
- [Include]共享头文件目录
- [Init]系统系统代码
- [Ipc]IPC通信,Message Queue等相关代码
- [Kernel]内核进程管理相关目录

Kernel 目录结构

- [Lib]基本库代码,标准C库的内核实现,memcpy等;
- [Mm]内存管理相关代码;
- [Net]网络核心代码;
- [Samples]内核模块模块的一些示例代码;
- [Scripts]编译内核的一些基本脚本;
- [Security]内核安全性代码;
- [Sound]音频处理的核心和音频驱动代码;
- [Tools]基本工具(amlogic专用);
- [Usr]制作文件系统代码;
- [Virt]虚拟机相关代码;

Kernel/ARCH/ARC目录

- ARC架构相关代码
- [Arch-apollo-h]:Apollo-h的相关基本硬件初始化;
- [Kernel]:中断,线程,硬件Cache等
- [Mm]:内存和MMU,Cache的硬件管理
- [Proc/arc700]:硬件启动代码,中断向量表,系统调用向量表;
- [Boot]:用来生成bootloader识别的Image;
- [BSP]:开发板相关目录的一个连接,在编译时生成

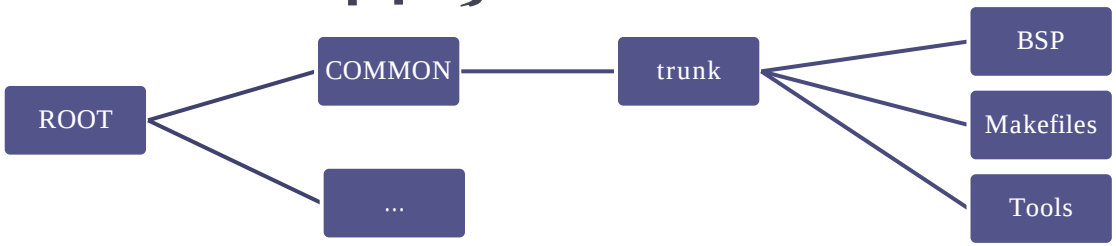
Kernel/include目录

- [asm-arc]:arc平台的头文件目录;
- [Asm-generic]:公共的硬件平台头文件;
- [Linux]:linux系统头文件;
- [Net]:网络相关头文件;
- [Sound]:声音相关头文件;
- [Asm/arc/arch]:Amlogic-arc芯片的头文件(驱动直接引用,它能根据芯片型号自动引用apollo或apollo-h的头文件);
- [Asm/arc/archapollo]:apollo系列头文件(为方便移植,不能直接引用);
- [Asm/arc/arch-apollo-h]:apollo-h系列的cpu头文件(为方便移植,不能直接引用);

Kernel/Driver目录

- 这个目录里面用来存放各种驱动;
- 我公司的驱动全部放在amloigc目录里面,请驱动开发按照这个结构存放,同时在Makefile里面的位置决定启动的顺序,非启动时间需要,请不要把驱动提前;
- [Kernel/Driver /amlogic]目录的基本驱动:
  - [Amports]:视频解码驱动;
  - [Audiodsp]:音频dsp的控制驱动;
  - [Cardreader]:sd,mmc等卡的驱动;
  - [Display]OSD,GE2D等驱动;
  - [HDMI]:HDMI接口驱动;
  - [I2C]:I2c驱动;
  - [Input]:输入设备驱动,现在主要有遥控设备驱动;
  - [Nand]:nand flash驱动;
  - [Net]:网络设备驱动,主要有以太网;
  - [Sound]:alsa音频驱动,(音频需要驱动启动后启动,这是 因为音频的核心在driver之后启动)
  - [Uart]:串口设备驱动,有硬件串口驱动和虚拟串口驱动(vuart),虚拟串口可以用来连接Metaware;
  - [USB]:USB的host控制器驱动;

Common目录



- [BSP]不同开发板对应的目录初始化目录,主要是pinmux,内存资源的分配;
  - 基本命名规则:芯片型号_内存配置
  - (如:7266_32x2)
- [Makefiles]编译系统的Makefiles,编译时会引用这些Makefile来进行编译;
  - [Makfile.common]:总控Makefile;
  - [Makfile.kernel]:编译内核使用的makefile;
  - [Makefile.busybox]:编译busybox使用的Makefile;
  - [Makefile.debug]:调式时使用的Makefile;
  - [Makefile.packages]:编译packages使用的makefile;(Makefile.common引用)
  - [packages.rules]:packages目录引用的Makefile,分析Config.in文件的配置,并对不同的packages编译进行管理;
  - [package.rules]:具体每个包可以引用的makefile,里面有基本的功能,可以减少重复的代码;
- [Tools]编译和生成Image时使用的工具和脚本;

Packages 目录

- Packages是软件包的集合
  - 包括了zlib, jpeg, freetype, directfb, microwindow, alsa-lib, amplayer等;
  - Config.in　　#p.menuconfig读取的包的配置文件,具体和busybox下的config.in相同;添加新的packages时需要修改;
  - Makefile　　#Config.in对应的Makefile,添加新的packages时需要修改;

Packages 目录

- 具体软件包的Makefile实现
- 以Zlib-1.2.3的 Makefile为例
  - P_FILE=zlib-1.2.3.tar.gz #软件包的文件名
  - FILE_DIR=zlib-1.2.3 #解压后的文件目录名
  - P_URL=http://www.zlib.net/zlib-1.2.3.tar.gz #下载该包的外部url地址;
  - P_GET_CMD=${HTTP_GET} #获取该包的命令,一般不需要修改;
  - P_GET_FLAGS=${HTTP_GET_FLAGS} #获取该包命令的使用参数;
  - TAR_CMD=${CMD_GZ} #解压包的命令,可以使用CMD_GZ或CMD_BZ2;也可以使用其他标准命令;
  - TAR_FLAGS=${CMD_GZ_FLAGS} #解压包命令使用的参数;和TAR_FLAGS对应;

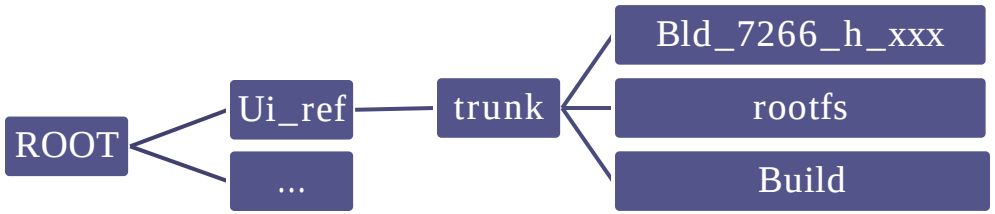
Packages 目录

- TRY_MAX=2 #如果获取不到该包是,最大重试次数
- DEPENDS= #他依赖的软件包,如果有多个软件包,只需要空格隔开
- PATCH_FILE = #这个包需要patch文件名,由于部分包直接编译无法生成有问题,需要打上Patch;一般patch文件就放在当前目录;
- CONFIG_FLAGS= --host=$(HOST_NAME) --prefix=${PREFIX} --with-softfloat --disable-python --disable-alisp --enable-shared --with-versioned=no \ --with-alsa-devdir=/dev
  #编译该包的命令选项,具体参考该包的help文件;
- P_FILE_F=${PKG_DIR}/${P_FILE}
- BUILD_DIR=${PKG_BUILD_DIR}/${FILE_DIR}
- .PHONY:all config before_cmd
- all:before_cmd config
- make -C ${BUILD_DIR} #编译
- make install #安装

Packages 目录

- install: #安装
- make -C ${BUILD_DIR} install
- .PHONY:configure
- configure:unzip_file #make configure
- cd ${BUILD_DIR}/&& \
- ./configure ${CONFIG_FLAGS}
- CONFIG_GEN_FILE=${BUILD_DIR}/config.mak #configure生成的文件
- CONFIG_DEP_FILE=${BUILD_DIR}/configure #config依赖文件
- UNZIP_GEN_FILE=${BUILD_DIR}/configure #解压生成文件
- UNZIP_DEP_FILE=${P_FILE_F} #解压依赖文件
- include $(MAKEFILES_DIR)/package.rules

Ui_ref目录



- 这个是参考的项目目录,他通过引用common,kernel,packages等目录来实现项目的管理;
- [env26.mk/env.mk]:用来配置环境,指定所引用的package,kernel,common和rootfs输出的目录;
- [bld_xxxx]:开发扳编译目录,所有编译命令都将在这个目录操作;
- [rootfs]:参考的rootfs目录,这个目录里面主要有用来参考的/etc/配置文件等;
- [build]:编译packages等文件的临时目录,用来存放编译出来的临时文件;

Env.mk的 设 置

- Sample:Ui_ref/trunk/env26.mk
- ROOT_DIR=${TOP_DIR}/../../.. #下面引用
- COMMON_DIR=${ROOT_DIR}/common #指定common目录
- ROOTFS_DIR=./rootfs #指定输出的rootfs目录.默认在bld_xxx目录里面,最好指定一个绝对目录,以方便调试,并且切换目录后不会存在问题;
- KERNEL_DIR= $(ROOT_DIR)/kernel_26 #指定内核原码目录
- BUSYBOX_DIR=${ROOT_DIR}/busybox #指定busybox原码目录
- PACKAGES_DIR=${ROOT_DIR}/packages #指定packages目录
- DEFAULT_BOOT_CMD="root=/dev/nfs nfsroot=10.68.11.57:/home/amlogicbj/rootfs/testfs rw noinitrd init=/init ip=10.68.11.72:10.68.11.1:10.68.11.1:255.255.255.0:target:eth0:off console=ttyS0,115200 mac=00:11:22:12:43:22" #指定内核默认的启动参数(没有使用uboot等loader引导时有效,调试用)
- PACKAGES_DIR_URL #指定packages包的地址;如果没有指定,会从openlinux.amlogic.com下载,通过指定能够加快下速度,方便内部开发使用;

Bld_XXXX目录的基本配置

- Makefile
  - TOP_DIR=${PWD}
  - SRC_DIR=${TOP_DIR}/../src
  - include ../env26.mk ##指定引用的环境设置文件目录;
  - BSP_DIR=${COMMON_DIR}/trunk/bsp/7266_h_64x2 ##指定开发板对应的bsp目录
  - RELEASE_DIR=${TOP_DIR}/build
  - BUSYBOX_CONFIG=${TOP_DIR}/busybox_config
  - KERNEL_CONFIG=${TOP_DIR}/kernel_config
  - PACKAGE_CONFIG=${TOP_DIR}/packages_config
  - START_INITRAMFS=no ##是否制作ramfs[yes/no],,如果制作,vmlinux里面就存在一个ramfs的;会在启动的时候自己加载到内存;
  - SRC_ROOT_FS=../rootfs/ ##rootfs源,主要存放着配置文件,编译busybox的时候会从这里copy到env.mk->ROOTFS指定的目录里面;
  - COMMON_SVN_PATH= ##源码对应的svn地址;
  - KERNEL_SVN_PATH=https://10.8.10.5/svn/Project_ARCLinux/trunk/kernel/v2.6.26
  - BUSYBOX_SVN_PATH=https://10.18.11.250/svn/model_linux/busybox/trunk
  - PACKAGES_SVN_PATH=https://10.18.11.250/svn/model_linux/packages
  - include ${COMMON_DIR}/trunk/Makefiles/Makefile.common ##引用主Makefile

Bld_XXXX目录的基本配置

- Wmake.bat:windows下面调试使用的批处理文件;
- Makefile: 项目编译目录,定义了具体引用的代码地址,板子相关设置;
- Kernel_config: 内核的配置文件;
- Busybox_config: busybox配置文件;
- Packages_config: packages配置文件;

主要编译命令

- 所有编译调试命令都在ui_ref/trunk/bld_xxx下面执行,具体哪个bld要根据您使用的板子决定;
- Make　　#编译全部,包括kernel, busybox, packages等;
- Make root #编译文件系统;
- Make yaffs　　#把文件系统打包生成yaffs文件系统的映像rootfs.yaffsimage,并保存在build目录;

主要编译命令

- 编译内核
  - Make k.menuconfig #配置内核,具体配置方法和直接配置内核一致,同时也可以使用make k.config, make k.xconfig等;
  - Make k #编译内核;
  - Make k.modules #编译内核模块,并自动安装到指定的ROOTFS/lib/modules目录;
  - Make k.clean #删除内核编译的临时文件,如果内核做了大的改动或更新,需要这么做,以防止播放部分没有重新编译导致问题;

主要编译命令

- 编译busybox
  - Make b.menuconfig　　#配置busybox
  - Make b　　#编译busybox
  - Make b.xxx　　#调用busybox内部的命令,包括clean,install;等

主要编译命令

- 编译packages(软件包)
  - Make p.menuconfig #配置软件包
  - Make p #编译软件包;
  - Make p.clean #删除编译生成的文件.
  - Make p.distclean #删除整个编译生成的packages目录;
  - Make p.${PACKAGES_NAME}.all #编译具体的软件包,这个 PACKAGES_NAME和软件包的目录名一致;

工具链

- Arc linux使用了两套工具链，分别用来编译内核和应用程序；
  - Arc-elf32-　　:基于new-lib的工具链，这个库不是基于linux系统调用开发的，主要用来编译非linux标准的应用程序；由于系统调用不一致，也不能用来编译应用程序；我们一般用来编译linux内核；
  - Arc-linux-uclibc- ：基于uclibc和linux的编译工具链，可以用来编译linux下的应用程序；

工具链

- Arc-elf32-gcc
- Arc-elf32-g++;
- Arc-linux-uclibc-gcc
- Arc-linux-uclibc-g++
  - 三个编译器都可以用来编译C和C++代码；
  - GCC一般用来编译c代码，同时也可以用来编译C++代码；
  - G++一般用来编译C++代码，编译C代码的使用会自动调用GCC；

工具链

- 其他主要工具，都省略了前缀，arc-elf32-和arc-linux-uclib-，如果不使用前缀，就是执行系统本身的工具链，工具的功能一样：
  - readelf #elf格式分析工具，可以查看section，符号表等；
  - Objdump #反编译工具，可以用来反编译elf格式文件，分析编译结果代码；
  - Objcopy #elf文件转换工具，可以用来生成bin文件，添加，删除symble，section等；
  - As #汇编代码编译工具
  - Ld #链接器，链接.o文件

工具链

- Ar #库制作工具，可以把多个.o文件制作为静态库，也可以把静态库分解为多个.o文件；
- Ranlib #跟新静态库的索引和符号表，加快静态库的访问速度，一般在ar后使用；
- Nm #elf文件符号表分析工具，linux下面的System.map就是使用nm来生成的；
- Strip #删除elf文件里面的symbols和sections；
- Size #显示elf程序中各段数据大小
- Run #简单的虚拟机，可以执行elf-gcc编译出来的简单文件；

工具链

- Gdb　　#gnu调试工具，能够单步，断点执行，可以分析堆栈，局部全局变量等；
- Insight　#gdb的图形版本，实际上是insight界面调用gdb来完成调试工作；
- Gcov　　#代码覆盖率分析；
- Gprof　　#代码执行次数统计工具，可以用来优化代码；

Thanks

Mail:Zhi.zhou@amlogic.com
msn:rising_o@msn.com
Skype:rising_o