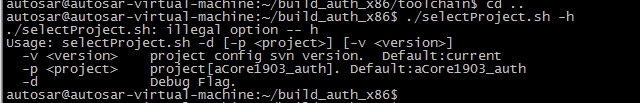
# 虚拟机创建

为了编译ara和app，需要一个虚拟机；可以先安装ubuntu16.04或者ubuntu18.04

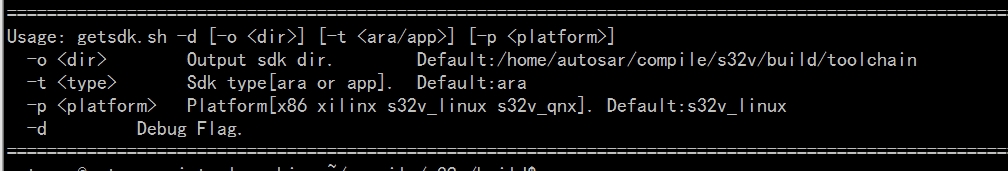
# 获取bsp的sdk

以aCore1903为例

1. svn co https://10.170.133.203/svn/SDC-EOS-aCore/SourceCode/compile/build
2. cd build
3. 如果对于1903\_auth，需要执行这一步 ./selectProject.sh –p aCore1903\_auth；对于本例中的aCore1903，不需要此步骤



1. ./getsdk.sh -p x86\_linux\_1903 参数意义如下



其中

-o 指定工具链安装的环境，如果不指定默认为当前目录下的toolchian

-t 指定sdk的类型， 值范围为 ara或app

（1）ara ：用于编译我们的底层模块， 脚本会将硬件供应商提供的原始工具链，和我们ara中的开源代码（boost，vsomeip等），放到 -o指定的目录中。

（2）app： 用于编译用户程序，脚本除了包含（1）中所有的文件外，还会把我们的ara 给用户环境释放的各类文件安装到 -o指定的目录中（即make release中sdk\_file中指定的那些文件）

-p用于指定平台。构建的时候，可以选择不同的平台，比如x86\_linux或者s32v\_linux

1. –p x86\_linux对应的是虚拟机版本的aCore平台；对于s32v2板子的linux版本，是-p s32v2\_linux\_1903，对于s32v2板子的qnx版本，是-p s32v2\_qnx\_1903

# 编译现有ara

source toolchain/environment.config

cd product/access/

如果之前source过别的工具链，则执行make distclean（这步会把makefile都删除了）

make platform.setup

make product.setup

make software（编译后的文件会安装到工具链的路径中） 或

make software release（编译后的文件会安装到）release/adaptive/output/sdk中

# ara中增加FC

## 修改mk文件

cd app/rules/

仿照其他mk，创建myApp.mk（注意目前大家都放在usr/目录下呢，参考别的mk就会发现这个现象）

vi app/Makefile

增加一行obj-y += myApp

## 编译

cd product/access

make software-myApp-clean

make software-myApp

## 成果物

编译出的东西在如下目录：

toolchain/sysroots/

## 安装

make release

## 成果物

位于product/access/release/adaptive/

# 生成sdk

vi product/access/config/build.config

将最后一行的BUILDCONFIG\_IS\_CREATE\_SDK=1

然后保存退出

make platform.setup

make product.setup

make software release

cd product/access/release/output/deploySdk/

./sdk\_s32v\_linux.sh –o xxx（其中xxx表示自己的sdk想要放置的位置）

# 编译app

进入上位机生成的代码目录output/code/，修改app代码

进入sdk目录，然后./compile\_app -s output/ -a all，其中output是代码目录，也可以是个绝对路径，并且不一定跟sdk在同一个目录下

注意-s后的参数所指明的目录下必须得是code和manifest目录，不要弄错了目录层级

编译后的东西在output目录下的release目录下面。

# 使用

## 安装iso

如果是使用制作好的iso文件（对应的是x86\_linux），则直接安装虚拟机，导入iso即可

安装iso的方法见相关说明文档

然后替换新编译的FC和app，运行即可；或者是重新出iso

## 挪文件

如果上位机工程在生成代码之前，配置了SoftwareCluster和SoftwarePackage，则可以使用UCM的方式（参考UCM培训文档），将output/upgrade/中的zip文件拷贝到运行环境，unzip之后出现个package.tgr，再tar –xf package.tgz –C /即可。

或者使用UCM客户端去安装、升级、卸载app（参考UCM培训文档）

如果没有配置SoftwareCluster和SoftwarePackage，则可以将output/release/目录下的东西scp –r到运行环境的根目录也行；如果要调试的是CM相关的，则需要注意/etc/vsomeip.json也得替换才行，并注意server和client的etc/vsomeip.json中的IP地址

# 调试em

在把自己的app上传之后，最好是看一下是否能够被em启动；这样可以防止重启后可能出现的卡住现象；可以下面这样：

execution-manager –DEBUG

如果可以打印各个进程启动成功或失败的信息，则说明机器清单校验通过了，重启也不会卡住了（顶多就是有一些进程无法启动，但是不会整个启动过程卡住）

需要注意的是，目前这个命令打印的一些错误信息并不会影响正常启动，这些错误信息可以忽略

# 启动其他FC和app

由于目前image中缺少sm的应用，没有去主动切换状态，因此em启动后会停留在MachineState.StartUp状态，就是说其他功能组的其他状态的app都没人去调用启动

最简单的解决方法是修改各自的执行清单，使之能在MachineState.StartUp状态中运行

正式使用应该是写一个sm应用去切换状态，这个应用可以是一个应用，也可以是多个应用；这些应用是与项目强相关的

# 注意事项

1. 如果终端出问题，乱打东西，用这个命令：stty sane
2. 如果重新make后ssh连不上，执行下面命令：rm -f /home/qq/.ssh/known\_hosts
3. 目前试验scp从外面向里面考不好使提示权限不足，从里面拷外面的东西好使；应该是由于在外面看不到scp用的里面的端口
4. ps aux查找qemu进程的pid的时候，不要grep qemu，会出现一大堆打印的；直接ps aux之后，自己找pid吧
5. make software-xxx之前，最好是先make software-xxx-clean一下，以保证会确实使用最新代码；尤其是修改CMakeList.txt，必须先clean，不然不会重新编译的