Practice With AOSP 14

01 Tháng Mười Hai 2024   12:28 CH

Prepare environment, source AOSP, source Kernel

Environment:

Install wsl2:

Run command:

```
wsl --install
wsl --set-default-version 2
```

Notice: Remember to enable Windows Subsystem for pLinux in window

```
Step 1: Open Windows Features Open the Start Menu and type "Windows Features" to find the "Turn Windows features on or off" option
Step 2: Enable "Windows Subsystem for Linux"
Step 3: Enable "Virtual Machine Platform"
Step 4: Restart Your Computer
```

Install Ubuntu20.04

```
wsl --install -d Ubuntu-22.04
```

Source AOSP

Pre-condition:

Source tree have to be setup in folder Ubuntu (e.g \\wsl.localhost\Ubuntu-20.04\home\truong\Automotive)

Install necessary package

```
sudo apt-get install git-core gnupg flex bison build-essential zip curl zlib1g-dev libc6-dev-i386 x11proto-core-dev libx11-dev lib32z1-dev libgl1-mesa-dev libxml2-utils xsltproc unzip fontconfig
```

Download source

Download repo init by manually - recommended (Ubuntu 20.04)

```
mkdir -p ~/.bin
PATH="~/.bin:${PATH}"
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/.bin/repo
chmod a+rx ~/.bin/repo
mkdir -p ~/.bin && PATH="~/.bin:${PATH}" && curl https://storage.googleapis.com/git-repo-downloads/repo > ~/.bin/repo && chmod a+rx ~/.bin/repo
```

Or install repo init by automatically (Ubuntu 22.04 or later)

```
sudo apt-get install repo
```

Create file sync.sh to do remain steps by automatically

```
repo init --partial-clone -b main -u https://android.googlesource.com/platform/manifest
repo sync -c -j16
```

Grant permission for sync.sh and run

```
chmod 777 sync.sh
./sync.sh
```

Notice:

Refer to https://source.android.com/docs/setup/reference/build-numbers#source-code-tags-and-builds change branch name

Sync source maybe failed due to some reasons: When sync completely, the log show which repo is failed. It just need to delete that repo and sync it again

For instance: repo fail: **platform/prebuilts/sdk**. Let remove below folders by command:

```
sudo rm -rf .repo/projects/prebuilts/sdk.git
sudo rm -rf .repo/project-objects/platform/prebuilts/sdk.git/
sudo rm -rf  prebuilts/sdk/
```

Then sync it again

```
repo sync -j1 -fail-past platform/prebuilts/sdk
```

Build source

Check swapsize, expected more than 32Gb

```
sudo swapon --show
```

Increase 32GB swapsize:

```
sudo fallocate -l 32G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
sudo cp /etc/fstab /etc/fstab.bak
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Run below command to build source

```
source build/envsetup.sh
lunch sdk_car_x86_64-trunk_staging-userdebug
make -j16
```

Run **lunch** to show all available target. Replace target before build image

sdk_car_x86_64-trunk_staging-userdebug is for AAOS run with Android Virtual Device using goldfish

Run command to run emulator

```
emulator -no-snapshot
```

Source Kernel

Need to select the kernel version that corresponds with the AOSP version. Refer to https://source.android.com/docs/devices/cuttlefish/kernel-dev

```
repo init -u https://android.googlesource.com/kernel/manifest -b common-android14-6.1
repo sync - -j16
```

Depending on the kernel version, there are two ways to build the kernel: using kleaf or build/build.sh

Build by kleaf

```
tools/bazel run //common-modules/virtual-device:virtual_device_x86_64_dist
```

Build by build/build.sh

```
BUILD_CONFIG=common-modules/virtual-device/build.config.virtual_device.x86_64 build/build.sh
```

Folder outs contain build result

Integrate customized kernel to AOSP

AOSP uses a prebuilt kernel to run, which is why AOSP is referred to as an OS based on a kernel.

We need to clone the kernel source for AOSP, build it independently, and then replace the custom kernel with the available prebuilt kernel in AOSP.

Depending on the target built before, it need to change difference build files. In this context, we are builting with x86_64 Android virtual device.

Modify this file: **device/generic/goldfish/board/kernel/x86_64.mk**

```
# Path to folder dist in customized kernel
KERNEL_ARTIFACTS_PATH := ../kernel/out/virtual_device_x86_64/dist
# Path to folder which includes file .ko
VIRTUAL_DEVICE_KERNEL_MODULES_PATH := $(KERNEL_ARTIFACTS_PATH)
# Path to prebuilt kernel
EMULATOR_KERNEL_FILE := $(KERNEL_ARTIFACTS_PATH)/bzImage
```

Re-build AOSP to apply changes then check version kernel before and after integrating

```
adb shell uname -r
```

Introduce demo work with AOSP

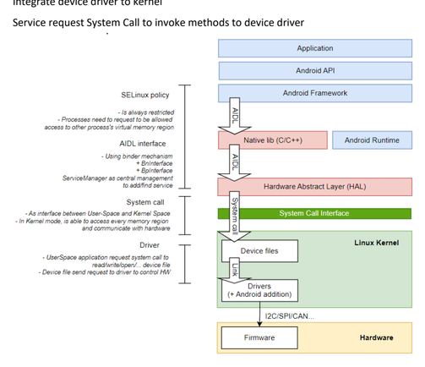
Introduce

Replace pre-built kernel in AOSP by customized kernel

Create App communicate with Service by AIDL

Integrate device driver to kernel

Service request System Call to invoke methods to device driver



Add native service

Prepare source tree

Create source tree as below which under folder hardware

```
example_service
  sepolicy\vendor
    exampleservice.te
    file_contexts
  service\com\service\exampleservice
    service.cpp
  Android.bp
  com.service.exampleservice.rc
  example_service.mk
```

Explaination

service.cpp

Include the main method. This function is called for the first time when the service is initialized

example_service.mk:

This file is declared so the system knows how to build our service. Depending on your target build, it needs to be integrated into the configuration file.

In this context, we are building with **sdk_car_x86_64.** So, it need to set config our service to file

```
device/generic/car/sdk_car_x86_64.mk
```

com.service.exampleservice.rc:

This file defines the initial actions to be performed before the service is initialized

Android.bp

This file defines the components included in our service. In this context, it specifies properties for the `cc_library`

file_contexts:

The name must match exactly. This file helps define the label for our service.

```
/vendor/bin/com.service.exampleservice        u:object_r:exampleservice_exec:s0
```

Exampleservice.te

This file defines the permissions for our service

- Define process as "domain"
- Permission to allow process init

Refer to commit "Initialize native service" for the code detail

Rebuild AOSP to apply the changes. Once the build is complete, check the output in `soong/.intermediates/hardware/example_service`

Add AIDL interface

Prepare source tree

Create source tree as below

```
example_service
  interfaces
    aidl
      com\service\api\aidl
        ITestServiceAidl.aidl
      Android.bp
    framework_compatibility_matrix.xml
  sepolicy\vendor
    exampleservice.te
    file_contexts
  service\com\service\exampleservice
    service.cpp
  Android.bp
  com.service.exampleservice.rc
  example_service.mk
```

Explaination

ITestServiceAidl.aidl

Define interface to communicate

The package name must match the file path (e.g., `package com.service.api.aidl` should correspond to the folder `com/service/api/aidl`

Android.bp:

This file defines the components included in our service. In this context, it specifies properties for the `aidl_interface`

framework_compatibility_matrix.xml

Define the AIDL component to expose it for client use.

This file can also be used for HIDL, so it is located under folder interfaces

Build and generate API

Run command under folder interfaces/aidl:

```
# Update api for system
m com.service.api.aidl-update-api
# Generate source
mm –j16
```

Check file .aidl be generated in **"current"** folder in your source tree

Check source gen in out/soong/.intermediate/hardware/example_service

```
example_service
  interfaces
    aidl
      aidl_api\com.service.api.aidl\current\com\...
        ITestServiceAidl.aidl
      com\service\api\aidl
        ITestServiceAidl.aidl
      Android.bp
    framework_compatibility_matrix.xml
  sepolicy\vendor
    exampleservice.te
    file_contexts
  service\com\service\exampleservice
    service.cpp
  Android.bp
  com.service.exampleservice.rc
  example_service.mk
```

```
example_service
  com.service.exampleservice\android_vend...
    obj
    unstripped
    com.service.exampleservice
    com.service.exampleservice.d
    meta_lic
    meta_lic.rsp
  interfaces\aidl
    com.service.api.aidl-api
    com.service.api.aidl-cpp-analyzer
    com.service.api.aidl-cpp-analyzer-source
    com.service.api.aidl-V1-cpp
    com.service.api.aidl-V1-cpp-source
    com.service.api.aidl-V1-ndk
    com.service.api.aidl-V1-ndk-source
```

Implement and register AIDL service

Add SEpolicy

Label for AIDL service in service_contexts

```
com.example.api.aidl.IExampleService/default       u:object_r:exampleaidl:s0
```

Define type for the label in exampleaidl.te

```
type exampleaidl, system_api_service, protected_service, service_manager_type;
```

Allow service to add AIDL service in exampleservice.te

```
allow exampleservice exampleaidl:service_manager { add };
```

Implement to add AIDL service using NDK backend

Implement folow the commit

I don't know but I tried to implement using CPP backend, it showed error

Build device driver

Prepare source tree

Prepare source tree as below

```
example_driver
  BUILD.bazel
  example_driver.c
  Makefile
```

Explaination

example_driver.c:

Simple device driver module which includes 2 main static methods:

static int __init m_init(void)

static void __exit m_exit(void)

BUILD.bazel:

Used to build by kleaf

Makefile:

Used to build by build/build.sh

-> add **visibility = ["//visibility:public"]** in common-modules/virtual-device/BUILD.bazel -> kernel_build - virtual_device_x86_64

Integrate our module to kernel and build

Add our module to build with kernel in \common-modules\virtual-device\BUILD.bazel

```
//common-modules/virtual-device/example_driver:example_driver
```

```
kernel_module_group(
    name = "virtual_device_x86_64_external_modules",
    srcs = [
        ":x86_64/goldfish_drivers/goldfish_address_space",
        ":x86_64/goldfish_drivers/goldfish_pipe",
        ":x86_64/goldfish_drivers/goldfish_sync",
        ":x86_64/v4l2loopback",
        ":x86_64/virtio_video",
        "//common-modules/virtual-device/example_driver:example_driver",
    ],
)
```

Add visibility to kernel build

```
visibility = [ "//common-modules/virtual-device/example_driver:__pkg__"],
```

```
kernel_build(
    name = "virtual_device_x86_64",
    srcs = [":virtual_device_x86_64_common_sources"],
    outs = [],
    arch = "x86_64",
    base_kernel = "//common:kernel_x86_64",
    build_config = "build.config.virtual_device.x86_64",
    make_goals = [
        "modules",
    ],
    module_outs = _VIRT_COMMON_MODULES + [
        # keep sorted
        "dummy-cpufreq.ko",
        "nd_virtio.ko",
        "test_meminit.ko",
        "virtio_pmem.ko",
    ],
    visibility = [
        "//common-modules/virtual-device/example_driver:__pkg__",
    ],
)
```

Rebuild kernel:

Check output file .ko if build successful

Implement driver code to register device file

Add source code as commit "Implement system call handler"

These logic will create device file which is mapping to our device driver

Check device file by running emulator and find file name "example_driver" in folder /dev

```
s –la /dev | grep example_driver
```

The FileOperation is used to mapping implemented logic with system call

```
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = driver_open,
    .read = driver_read,
    .write = driver_write,
    .release = driver_release,
    .llseek = driver_llseek,
    .mmap = driver_mmap,
    .unlocked_ioctl = driver_ioctl,
    .poll = driver_poll,
    .fsync = driver_fsync,
    .fasync = driver_fasync,
};
```

Add application to AOSP

Prepare source tree

Prepare source tree as below:

```
app
  res
    layout
      activity_main.xml
    mipmap
      ic_launcher.png
  src\com\client\exampleapp
    ExampleApp.java
  Android.bp
  AndroidManifest.xml
```

Explaination

activity_main.xml:

code xml to create UI

ExampleApp.java

Extends from activity to implement logic app

Android.bp

Define necessary components for application. Remember add lib **"com.example.api.aidl-V1-java"** to use AIDL

```
static_libs: [
    ...
    "com.example.api.aidl-V1-java",
],
```

AndroidManifest.xml:

Define Application for system

Add product package of app to system build

Add package name to example_device.mk

```
PRODUCT_PACKAGES += \
        ExampleApp \
PRODUCT_ARTIFACT_PATH_REQUIREMENT_ALLOWED_LIST += \
        system/app/ExampleApp/%
```

Modify SEpolicy

Label for app instance seapp_contexts

```
user=_app seinfo=platform name=ExampleApp domain=exampleapp type=app_data_file levelFrom=user
```

Create file exampleapp.te to grant permission for application

```
# Define application process as domain
type exampleapp, domain;
# Permission to allow application transfer data or call method to native service
allow platform_app exampleservice:binder { call transfer };
```

Implement system call logic

Implement code as commit

Commit "Implement system call logic"

Add SEpolicy

Define label for /dev/example_driver in **file_contexts**

```
/dev/example_driver                    u:object_r:example_driver_exec:s0
```

Define dev_type for example_driver_exec in **exampleservice.te**

```
type example_driver_exec, dev_type;
```

Allow permission reading/writing device file in **exampleservice.te**

```
allow exampleservice example_driver_exec:chr_file rw_file_perms;
```

Implement logic to work with AIDL service- from application

Get service and invoke methods

Using name of AIDL service to get Binder from ServiceManager

```
final String instance = "com.example.api.aidl.IExampleService/default";
 IBinder binder = ServiceManager.getService(instance);
IExampleService mExampleService = IExampleService.Stub.asInterface(binder);
```

Invoke methods by using mExampleService

The code must be inner try catch block to handle exception "RemoteException" (due to call method using IPC)

```
try {
    mExampleService.registerListener(mExampleListener);
    Log.d(TAG, "Call method unregisterListener successfully");
} catch (android.os.RemoteException e) {
    Log.e(TAG, "Call method unregisterListener failed", e);
}
```

Implement ExampleServiceListener to receive data

Create ExampleServiceListenerFromApp.java which is implemented from **IExampleListener.Stub**

```
class ExampleListenerFromApp extends IExampleListener.Stub
```

Override methods: **notifyEvent**, getInterfaceVersion, getInterfaceHas

The notifyEvent method is used to receive data transfers from the native service.

```
@Override
public void notifyEvent(int number)
{
    Log.d(TAG, "notifyEvent is called with number = " + number);
}
```