Using IPC in an Android app

Contents

Using IPC 3.x in a Google Play App

IPC (http://processors.wiki.ti.com/index.php/IPC_Users_Guide/About_IPC) 3.x has built-in Android support that allows it to be built and used in 'system applications' in the Android source tree. This topic explores what needs to be done in order for native apps developed using the Google NDK to take advantage of services provided by IPC. We'll take a look at how to implement the host-side code of the IPC example ex02_messageq in the context of a native Android App.

In our procedure we used a DRA7XX development board, but technically this can be done on any device that is supported by IPC.

Prerequisites

Obtain a DRA7XX development board

Download and install the following packages:

- Android SDK/ADT bundle (https://developer.android.com/sdk/installing/bundle.html)
- Android NDK (https://developer.android.com/tools/sdk/ndk/index.html)
- TI Android source tree release 6AK.1.0 (http://omappedia.org/wiki/6AK.1.0_Release_Notes)
- IPC 3.x for Android (http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/index.html)

Follow the directions in the IPC Install Guide (http://ap-fpdsp-swapps.dal.design.ti.com/index.php/IPC_Install_Guide_Android) to install and rebuild IPC under the 'hardware/ti' directory as instructed.

You should then follow the instructions in the TI Android source tree release to create all the images and flash them onto your DRA7xx board. You may want to try running an IPC test or example afterwards to ensure your setup is fully functional. The Android 'adb' utility needs to be used for this procedure, so make sure your development board is connected to your host development machine via a USB cable.

Procedure

Update slave executable on the target

Rebuild the ex02_messageq example in IPC (refer to IPC documentation on how to do this), after updating 'ex02_messageq/makefile' with a reduced PROCLIST that only contains the list of the slaves for which we want to build an image. For example, modify it as follow if we just want to build the image for IPU2:

```
# edit PROCLIST list to control how many executables to build
PROCLIST = ipu2
```

Boot up your development board. Copy the slave image from your development machine to your target's '/vendor/firmware' directory using adb:

```
dev host# adb push <IPC INSTALL DIR>/examples/ex02_messageq/ipu2/bin/debug/server_ipu2.xem4 /vendor/firmware/dra7-ipu2-fw.xem4
```

Create an APK file with an application that invokes IPC

Import and rebuild the sample NDK application described in the section "Exploring the native-activity Sample Application" on the Android NDK page (https://developer.android.com/tools/sdk/ndk/index.html).

Copy the 'ex02_messageq/host/shared' folder into the '<workspace_path>/NativeActivity' directory

Copy 'ex02_messageq/host/App.c' and 'ex02_messageq/host/App.h' into the '<workspace_path>/NativeActivity/jni' directory

Modify 'main.c' in the NativeActivity project's jni directory to use IPC:

<syntaxhighlight lang=c> /* ... */

1. include <android_native_app_glue.h>

/* package header files */

1. include <ti/ipc/Std.h>
2. include <ti/ipc/Ipc.h>

1. include <ti/ipc/MultiProc.h>

/* local header files */

1. include "App.h"

/* Which slave to talk to */ static String Main_remoteProcName = "IPU2";

1. define LOGI(...) ((void)__android_log_print(ANDROID_LOG_INFO, "native-activity", __VA_ARGS__))
2. define LOGW(...) ((void)__android_log_print(ANDROID_LOG_WARN, "native-activity", __VA_ARGS__))

/* ... */

void android_main(struct android_app* state) {

```
  struct engine engine;
  UInt16      remoteProcId;
  Int         status = 0;
```

/* ... */

```
  if (state->savedState != NULL) {
      // We are starting with a previous saved state; restore from it.
      engine.state = *(struct saved_state*)state->savedState;
  }
```

```
  /* Ipc initialization */
  status = Ipc_start();
  if (status >= 0) {
      /* application create, exec, delete */
      remoteProcId = MultiProc_getId(Main_remoteProcName);
```

```
      /* application create phase */
      status = App_create(remoteProcId);
```

```
      if (status < 0) {
       LOGI("App_create failed: status = %d\n", status);
       return;
      }
```

```
      /* application execute phase */
      status = App_exec();
```

```
      if (status < 0) {
       LOGI("App_exec failed: status = %d\n", status);
       return;
      }
```

```
      /* application delete phase */
      status = App_delete();
```

```
      if (status < 0) {
       LOGI("App_delete failed: status = %d\n", status);
          return;
      }
```

```
      /* Ipc finalization */
      Ipc_stop();
```

```
       LOGI("Application run was successful!!!!\n");
```

```
  }
  else {
       LOGI("Ipc_start failed: status = %d\n", status);
      return;
  }
```

```
  // loop waiting for stuff to do.
```

/* ... */ </syntaxhighlight>

Modify '<workspace_path>/NativeActivity/jni/Android.mk' as follow, by setting the AFS_PATH to the location where your Android source tree is installed:

<syntaxhighlight lang='make'> LOCAL_PATH := $(call my-dir)

1. Path to Android source tree/filesystem

AFS_PATH := /db/builds/vw/6AK.1.0/mydroid

1. Path to IPC installation directory

IPC_ROOT := $(AFS_PATH)/hardware/ti/ipc/ipc_3_22_00_03_eng

1. Path to IPC shared libraries (.so)

```
LIB_PATH += $(AFS_PATH)/out/target/product/jacinto6evm/system/lib

include $(CLEAR_VARS)

LOCAL_MODULE  := libtiipcutils LOCAL_SRC_FILES := $(LIB_PATH)/libtiipcutils.so

include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)

LOCAL_MODULE  := libtiipc LOCAL_SRC_FILES := $(LIB_PATH)/libtiipc.so

include $(PREBUILT_SHARED_LIBRARY)


include $(CLEAR_VARS)

LOCAL_C_INCLUDES += $(IPC_ROOT)/linux/include \

                  $(IPC_ROOT)/packages \
                  $(IPC_ROOT)/hlos_common/include

LOCAL_MODULE  := native-activity LOCAL_SRC_FILES := main.c App.c LOCAL_LDLIBS  := -llog -landroid -lEGL -lGLESv1_CM -lc

LOCAL_STATIC_LIBRARIES := android_native_app_glue

LOCAL_SHARED_LIBRARIES := \

  libtiipcutils libtiipc

include $(BUILD_SHARED_LIBRARY)

$(call import-module,android/native_app_glue) </syntaxhighlight>
```

In a terminal window on your Ubuntu development machine, go into the '<workspace_path>/NativeActivity' directory. Run

```
dev host# <path to ndk>/ndk-build
```

This invokes the NDK to rebuild the native application, and produces a shared library 'libnative-activity.so'.

In order to produce an APK file for this application, you should have the Android project opened in Eclipse (the instructions on the NDK page should have taught you how to do so). Then follow these steps:

- Right-click on the NativeActivity project in the Package Explorer. Select Export...
- Select Android->Export Android Application. Hit Next
- The Project Checks screen should say no errors found. Hit Next
- Either create a new keystore or use an existing keystore if you already have one. Hit Next
- Either create a new key or use an existing key if you already have one. Hit Next
- The path to the generated APK file will be shown. Hit Finish

Now you have generated an APK file for your application.

Install the app and run it

After you have generated the APK file, the next step is to install it to the target. Boot up your board with Android if you haven't already done so. On your development host machine, install your APK file:

```
dev host# adb install -r NativeActivity.apk
```

Start the adb shell as root:

```
dev host# adb root
dev host# adb remount
dev host# adb shell
```

In the shell, launch the IPC LAD process

```
adb shell# /system/bin/lad_dra7xx -l lad.txt
```

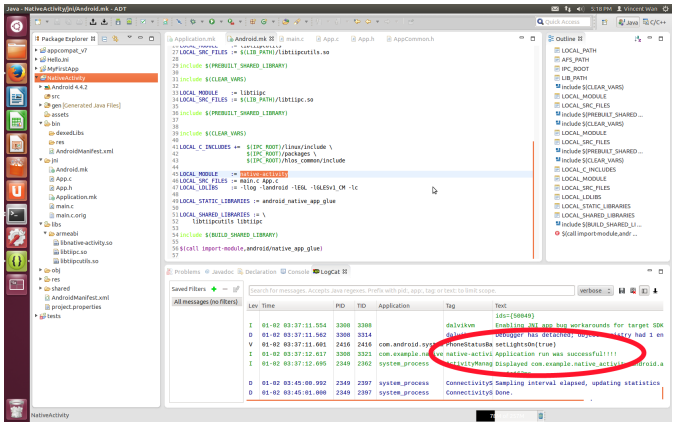
Modify the permissions on the command pipe created by LAD to make it accessible by all users:

```
adb shell# cd /data
adb shell# chmod 777 lad
adb shell# chmod 777 lad/LAD
adb shell# chmod 777 lad/LAD/LADCMDS
```

Launch the app:

```
adb shell# am start -a android.intent.action.MAIN -n com.example.native_activity/android.app.NativeActivity
```

If you have Eclipse open, you should be able to see a success message in the LogCat window:



You can also verify that the slave has received all messages and replied to them by looking at the remote log:

```
adb shell# cat /d/remoteproc/remoteproc1/trace0
```

The expected output on the remote processor should be similar to this:

```
[0][  14154.495] [t=0x00000010:a2a61aa9] Server: --> Server_exec:
[0][  15325.319] [t=0x00000012:02c946d7] Server: Server_exec: processed cmd=0x0
[0][  15325.319] [t=0x00000012:02cacc5f] Server: Server_exec: processed cmd=0x0
[0][  15325.320] [t=0x00000012:02cc96df] Server: Server_exec: processed cmd=0x0
[0][  15325.320] [t=0x00000012:02ce6e07] Server: Server_exec: processed cmd=0x0
[0][  15325.320] [t=0x00000012:02d01ec7] Server: Server_exec: processed cmd=0x0
[0][  15325.320] [t=0x00000012:02d1d3bf] Server: Server_exec: processed cmd=0x0
[0][  15325.321] [t=0x00000012:02d3a1e5] Server: Server_exec: processed cmd=0x0
[0][  15325.321] [t=0x00000012:02d5625f] Server: Server_exec: processed cmd=0x0
[0][  15325.321] [t=0x00000012:02d72495] Server: Server_exec: processed cmd=0x0
[0][  15325.322] [t=0x00000012:02d8f813] Server: Server_exec: processed cmd=0x0
[0][  15325.322] [t=0x00000012:02daba7f] Server: Server_exec: processed cmd=0x0
[0][  15325.322] [t=0x00000012:02dc7fef] Server: Server_exec: processed cmd=0x0
[0][  15325.323] [t=0x00000012:02de54a1] Server: Server_exec: processed cmd=0x0
[0][  15325.323] [t=0x00000012:02e01bd1] Server: Server_exec: processed cmd=0x0
[0][  15325.323] [t=0x00000012:02e164c3] Server: Server_exec: processed cmd=0x2000000
[0][  15325.323] [t=0x00000012:02e2b5b9] Server: <-- Server_exec: 0
[0][  15325.324] [t=0x00000012:02e3a5b9] Server: --> Server_delete:
[0][  15325.324] [t=0x00000012:02e55c91] Server: <-- Server_delete: 0
[0][  15325.324] [t=0x00000012:02e7f80b] Server: Server_create: Slave is ready
[0][  15325.325] [t=0x00000012:02e92591] Server: <-- Server_create: 0
[0][  15325.325] [t=0x00000012:02ea30f7] Server: --> Server_exec:
```

Reference code

For reference purposes, the project files are available here: File:NativeActivity.tar.gz.

{{

1. switchcategory:MultiCore=
   - For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
   - For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

   Please post only comments related to the article **Using IPC in an Android app** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Using IPC in an Android app** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **Using IPC in an Android app** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **Using IPC in an Android app** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **Using IPC in an Android app** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Using IPC in an Android app** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Using IPC in an Android app** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **Using IPC in an Android app** here.*

*For technical s please post yo questions at http://e2e.ti.cor Please post on comments abo article **Using I an Android ap***

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

Retrieved from "https://processors.wiki.ti.com/index.php?title=Using_IPC_in_an_Android_app&oldid=188336"

This page was last edited on 15 December 2014, at 00:44.

Content is available under Creative Commons Attribution-ShareAlike unless otherwise noted.