Building native Android apps locally using Python

Subhodeep Chakraborty
Department of Electrical Engineering
IIT Hyderabad
Email: ee25btech11055@iith.ac.in

I. Motivation

Today, an Android phone is functionally equivalent to a development workstation for most purposes. There are very few cases where there is not a way to achieve the required goals using a mobile device.

Despite this, traditionally the *building* of native Android apps itself has been restricted to the desktop. This is due to the large SDK and NDK sizes, and also long build times. Some recent developments, however, have made change possible.

This guide aims to showcase some work done in the open source community in the past few months that now allows for the development, building and running of native apps directly from the phone itself, without the need of any other device.

II. Setup

1) Follow the instructions in the follwing repo to setup proot-distro and termux. Following instructions are to be executed inside proot-distro.
   https://github.com/gadepall/fwc-1

2) Install the builder package and dependencies using pip (Python version 3.8 to 3.10)
   pip install python-for-android Cython==3.0.0

3) Install dependencies.
   apt-get update
   apt-get install -y ant autoconf automake ccache cmake g++ gcc git lbzip2 libffi-dev libltdl-dev libtool libssl-dev make openjdk-17-jdk patch pkg-config python3 python3-dev python3-pip python3-venv sudo unzip wget zip

4) The normal Android SDK and NDK provided by Google cater to Linux users, and does not work correctly on Android.

5) Download the NDK patched for Termux use by users codehasan, Party233 and Lzhiyong on Github:
   wget https://github.com/codehasan/dex2c/releases/download/ollvm-termux/android-ndk-r25c-ollvm-aarch64.tar.xz

6) For the SDK, go to the following link and scroll down to "Command line tools only" and download the Linux version.
   https://developer.android.com/studio

7) Extract both the files and place in appropriate directory structure (Assuming both files are in correct direction. Replace XX with correct version in file name).

```
tar -xf android-ndk-r25c-ollvm-aarch64.tar.xz
mv XX.XX android-ndk-r25c
ln -s android-ndk-r25c/toolchains/llvm/prebuilt/linux-aarch64 android-ndk-r25c/toolchains/llvm/prebuilt/linux-x86_64
mkdir sdk
unzip commandlinetools-linux-XX_latest.zip
mv cmdline-tools sdk/tools
```

8) Download env.sh from SubhoBoy/buildterm Github repo and run it.

```
wget https://raw.githubusercontent.com/SubhoBoy/buildterm/refs/heads/main/env.sh
chmod +x env.sh
source ./env.sh
```

9) Now we install the full SDK. We use Android API level 27 for compatibility with the older Python version we are using. Use the list command to find latest build tools version and install it.

```
cd sdk
sdkmanager "platforms;android-27"
sdkmanager --list | grep build-tools
sdkmanager "build-tools;XX.XX.XX"
cd ..
```

10) Finally, we need to set up the Android Asset Packaging Tool for use on Termux:

```
cd ~
wget https://github.com/rendiix/termux-aapt/raw/main/prebuilt-binary-android-12%2B/arm64/aapt2
chmod +x aapt2
mkdir -p ~/.gradle
echo "android.aapt2FromMavenOverride=$(readlink -f ~/aapt2)" >> ~/.gradle/gradle.properties
cd -
```

III. Compilation

1) The syntax for the compilation command is as follows:

```
p4a apk --private $HOME/code/myapp
  --package=org.example.myapp --name
  "My application" --version 0.1
  --bootstrap=sdl2
  --requirements=python3,kivy,libffi
  --arch=arm64-v8a

```

Here, `--private` is the directory containing the app's `main.py`. `--requirements` should include all the packages needed by the app.

2) Navigate to your app's folder. For example, using calculator repo for EE1003:

```
git clone https://github.com/SubhoBoy/
   calculator
cd calculator
```

3) Run the above command as per your app's details. For example:

```
p4a apk --private
   /root/calculator/android
   --package=org.subhoboy.calci
   --name "Calci" --version 0.1
   --bootstrap=sdl2
   --requirements=python3,kivy,libffi
   --arch=arm64-v8a
```

On first run, this command will take roughly 10-15 minutes to run with a decent internet connection. It will result in an error regarding `libffi.so`.

4) Find location of system `libffi.so`:

```
find / -name libffi.so
```

5) Copy system libffi.so to our build folder.

```
cp /usr/lib/aarch64-linux-gnu/libffi.
   so $HOME/.local/share/python-for-
   android/build/other_builds/libffi/
   arm64-v8a__ndk_target_21/libffi/.
   libs/libffi.so
```

(Edit paths for your system, if needed)

6) Run the command given in step 2 again. This time, it should finish compiling successfully in under a minute. You will find a `.apk` file generated in the current directory.

```
mv unnamed_dist_2-debug-0.1.apk
   /sdcard/calci.apk
```

This `.apk` file may be installed on the device. Note that due to being compiled for an older version of Android and being unsigned, it may be flagged as untrusted by Google Play; this can be safely bypassed.

IV. Acknowledgements

The preparation of this guide was made possible only by the community of open source developers on Github. Specifically, it uses code patched by Github users codehasan, Party233, Lzhiyong, rendiix and the kivy team.

I am grateful to Dr. G.V.V. Sharma for giving me the motivation to work on this niche but high-potential field. Finally, I would like to thank my friend Mr. Vivek K. Kumar for his constant support through the research and debugging phase.