# COM1032 Operating Systems
# Lab 2
## OS Commands in Linux Kernel in Android Devices

# Purpose:

The purpose of this lab session is to familiarise yourself with the Linux Operating System Kernels in different devices. This document focuses on Android Devices.

# Aim

By the end of the lab you will be able to:

- Setup and connect your android device to your PC using the Android Debug Bridge (ADB) to query some of the OS processes of Android.

# Android ADB Commands

Connect your Android device to a lab machine using a USB cable and then run the following commands:

To restart in USB mode, in your device first go to Settings -> System -> Development option -> enable USB debugging:

```
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
```

If you do not find any devices listed, try to connect wirelessly, by following the steps below:

1. Type `adb tcpip 5555`.
2. On your Android device, go to Settings -> System -> Development option -> USB debugging --> Uncheck it for TCPIP connection.
3. Go to Settings -> Wireless & networks -> Wi-Fi -> Click the settings icon to find out the device IP Address.
4. Type `adb connect 192.168.1.2` if this is your device IP address.

```
$ adb devices -l
List of devices attached
G5NPCX0207036UZ        device product:WW_P00A model:P00A device:P00A_2
```

**Exercise 2: How do you kill this server that you started?**

# HAL commands:

**Exercise 3: a) What is the processor type and capabilities of your device?**

```
$ adb shell cat /proc/cpuinfo
$ adb shell getprop ro.product.cpu.abilist
```

**Exercise 3: b) arm64-v8a,armeabi-v7a,armeabi : what do these mean?**

**Hints: https://developer.android.com/ndk/guides/abis**

```
$ adb reboot
```

You can specify that it reboots to the bootloader.

```
$ adb reboot-bootloader
```

When you're working inside the bootloader, adb no longer works. You're not yet booted into Android, and the debugging tools aren't active to communicate with. You'll need to use the fastboot command in its place. Fastboot is probably the most powerful Android debug tool available, and many devices don't have it enabled. If yours does, you need to be sure things are communicating. That's where the fastboot devices command comes into play. At the prompt, just type in fastboot devices and you should see a serial number, just like the adb devices command we looked at earlier.

```
$ fastboot devices
$ fastboot –help
$ fastboot reboot
```

**Exercise 4: What you can do to your device using fastboot commands? How useful is the following command:** `fastboot flashing unlock`

Most devices can also boot into the recovery directly

```
$ adb reboot recovery
```

When you're working inside the bootloader, adb no longer works. You're not yet booted into Android, and the debugging tools aren't active to communicate with. You'll need to use the `fastboot` command in its place if it is available in your device.

Restarts the `adbd` daemon with root permissions

```
$ adb root
```

Example output to the last command is as follows:

```
error: device unauthorized.
```

```
This adbd's $ADB_VENDOR_KEYS is not set; try 'adb kill-server' if that seems wrong.
Otherwise check for a confirmation dialog on your device.
$ adb root
adbd cannot run as root in production builds
$ adb run-as package_name
```

The last command enables you to run commands on a device as an app (specified using package_name). This lets you run commands in adb as if the app you specify is running the command (that is, you have the same device access that the app has), without requiring root access. This might be necessary when using adb on a non-rooted device or an emulator with a Play Store image. The app must be debuggable.

# File Management:

**OS manages available storage media, files and directories:**

To copy a file onto your Android device programmatically, you want to use the adb push command:

```
$ adb push music_file /sdcard/Music
```

Assuming music_file exists in the current folder, and /sdcard/Music is the music folder on your Android device. Generally the push command takes two parameters: the full path of the file you're pushing, and the full path to where you want to put it.

To copy a file from your Android device programmatically, you want to use the adb pull command:

```
$ adb pull /sdcard/demo.mp4 ./
```

Assuming /sdcard/demo.mp4 exists in the mentioned path in the device, and moved to the current directory on the host machine.

There are two ways to use the adb shell command, one where you send a command to the device to run in its own command line shell, and one where you actually enter the device's command shell from your terminal.

list of all the available shell programs, use the following command:

```
$ adb shell ls /system/bin
```

# Activity Manager:

The activity manager (am) tool performs various system actions, such as start an activity, force-stop a process, broadcast an intent, modify the device screen properties, and more.

```
$ adb shell am start -a android.intent.action.VIEW
```

# Package Management:

The package manager (pm) tool performs actions and queries on app packages installed on the device. To list the packages in the device:

```
$ adb shell pm list packages
```

To install a package in the device:

```
$ adb shell pm install path_to_apk
```

You can use the -t option with the install command when you install a test APK. If you're updating an app, you use the -r switch: `adb install -r TheAppName.apk`. There is also a -s switch which tries to install on the SD card

**Exercise 5: How do you uninstall a package? What else you can do with pm?**

# Policy Manager:

The device policy manager (dpm) tool controls the active admin app or change a policy's status data on the device. The following command sets the current user as the active admin

```
$ adb shell dpm set-active-admin --user current
```

# Process Management:

Check running processes: Let's say we want to check if com.android.phone is running:

```
$ adb shell ps m.android.phone
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
radio     1389  277   515960 33964 ffffffff 4024c270 S com.android.phone
$ adb shell pidof com.android.phone
```

**UNIX Pipes**

```
$adb shell ps | grep apps | awk '{print $9}'
```

If you want to directly get the package name of the current app in focus, use this adb command:

```
$ adb shell dumpsys window windows | grep -E 'mFocusedApp'| cut -d / -f 1 | cut -d " " -f 7
```

Top is available in adb, so you can do things like the following command to monitor the top five CPU hogging processes.

```
$ adb shell top -m 5
```

Or to record this for one minute and collect the output to a file on your computer:

```
$ adb shell top -m 5 -s cpu -n 20 |tee top.log
```

To show the process priorities use the –p option for ps for all threads (-t):

```
$ adb shell ps -t –p
"WindowManager" prio=5 tid=20 NATIVE
  | group="main" sCount=1 dsCount=0 obj=0x409404d8 self=0x264970
  | sysTid=2884 nice=-4 sched=0/0 cgrp=default handle=2510248
  | schedstat=( 4116000 5332000 12 ) utm=0 stm=0 core=0
```

Generally the ps command takes the following options:

- `-t` show threads, comes up with threads in the list
- `-x` shows time, user time and system time in seconds
- `-P` show scheduling policy, either `bg` or `fg` are common, but also `un` and `er` for failures to get policy
- `-p` show priorities, niceness level
- `-c` show CPU (may not be available prior to Android 4.x) involved
- `[pid]` filter by PID if numeric, or…
- `[name]` …filter by process name

# Log Files Accessing:

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the `Log` class.

```
$ adb logcat
```

Logcat will print all logs, you will need to filter to find what you need.

**Exercise 6: How do you filter for errors only? What else you can filter on?**

**Hint: https://developer.android.com/studio/command-line/logcat**

# For complete reference:

https://developer.android.com/studio/command-line/adb

http://adbshell.com/

https://www.all-things-android.com/content/android-toolbox-command-reference