ITMO Advanced OS 2024

Aleksei Romanovskii, PhD ,
SPb Research Center (CBG OS Lab)
Lesson 2022.12.25

Content

- Basics of mobile OS programming (AARCH64, Android)
- How to uninstall SW bloat from mobile device
- How to cross-compile an app for mobile device
- How to put any files on (get any files from) mobile OS FS
- How to run the app on mobile device
- How to measure instructions and cycles spent by the app code
  - using perf
  - using sys calls
- Ideas for talks/proposals to go to premium exam (get +1 for exam mark)

Basics of mobile OS programming

- Goal: illustrate some concepts previously covered in the lessons
  - not related to the course practice
  - no kernel programming so far – user-space only
  - no mobile kernel builds (still this may be an idea for a talk/PPT to prepare)

- Pre-requisites: Android device, Linux machine and some knowledge of bash/terminal/CLI
  - no Android Studio needed, will use ADB
  - may also be done on Windows
  - may be done for Harmony OS (an idea for a talk/PPT to prepare)

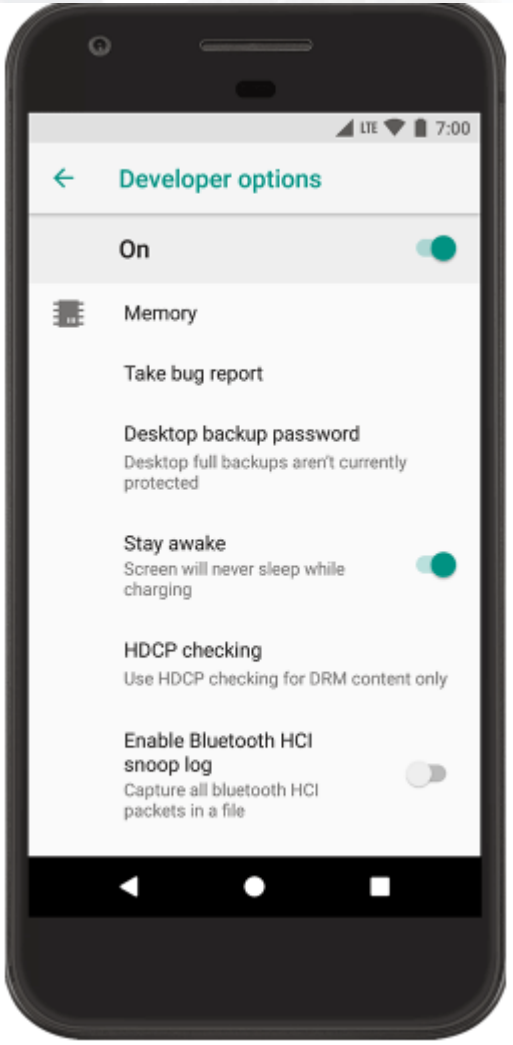
- Safety guaranteed – no damage to your Android device

Basics: ADB

- Android Debug Bridge (adb) is a command-line tool that lets you communicate with a device.
- The adb command facilitates a variety of device actions, such as installing and debugging apps, exchanging files with Android device, etc..
- The adb provides access to a Linux shell that you can use to run a variety of commands on a device.
- It is a client-server program that includes three components:
  - A client, which sends commands to device daemon. The client runs on Linux machine. Invoked from a command-line by issuing an adb command.
  - A server, which manages communication between the client and daemon on device. The server runs as a background process on Linux machine.
  - A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device (pre-installed).

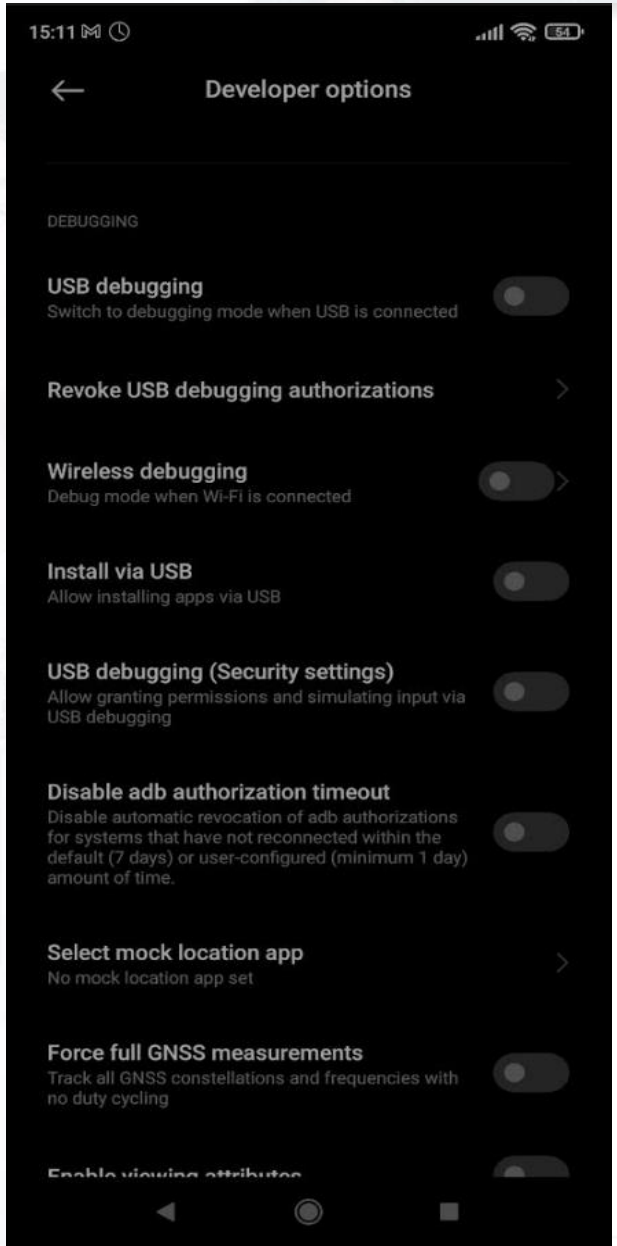
Basics: Prepare Android device for adb

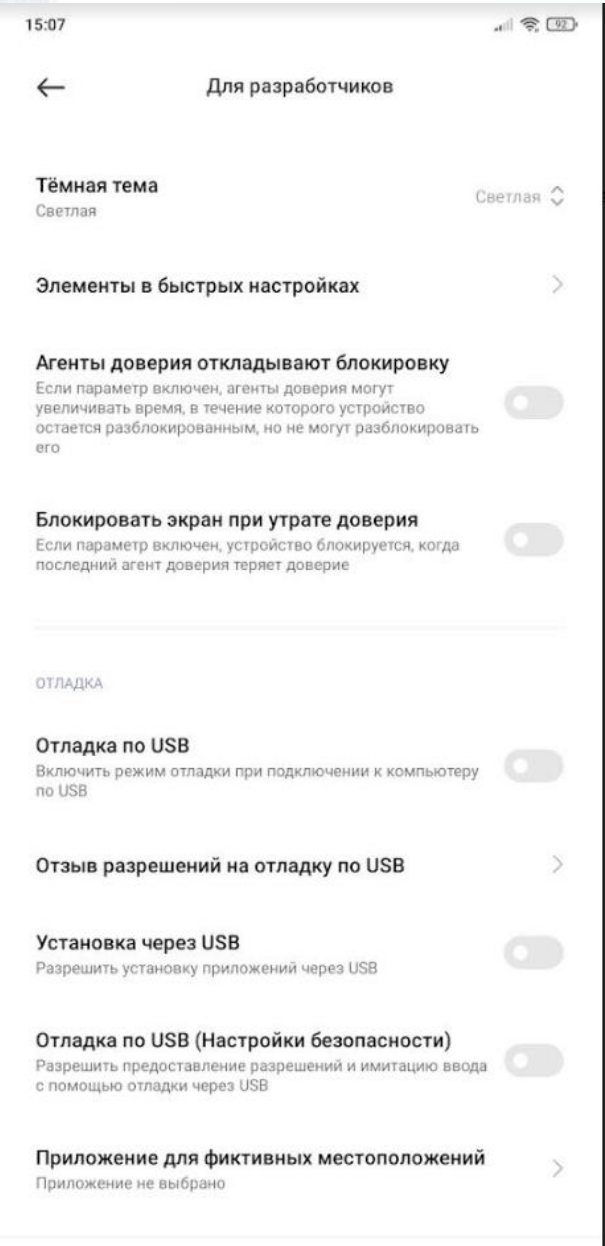
- Enable “Developer options” on device:
- Android 9 (API level 28) and higher: go to Settings > About Phone > Build Number. Tap 7 times on Build Number
- Android 8.0.0 (API level 26) and Android 8.1.0 (API level 26): Settings > System > About Phone > Build Number. Tap 7 times
- Android 7.1 (API level 25) and lower: Settings > About Phone > Build Number. Tap 7 times
- Find Developer Options (Режим Разработчика) in Settings, go there.
- At the top of the Developer options screen, one can toggle the options on and off.



Basics: Prepare Android device for adb

- At the Developer options screen find USB debugging
- Toggle it On
- Do not forget to toggle it Off after finishing experiments





Basics: Prepare Android device for adb

- Later on, when you connect your phone to your computer, you'll see a popup entitled "Allow USB Debugging?" on your phone.

- Check the "Always allow from this computer" box (optional) and tap OK.

Basics: ADB installation and running the client

- Debian/Ubuntu type in terminal:
  - *sudo apt install adb*
- Connect the mobile device to Linux machine and allow debugging (previous slide)
- Type in terminal
  - *adb devices*
- The output to terminal, among other information, should show device ID, like this
  - *1C161FDF600KTM device*
- When you start adb client (e.g. “adb devices”):
  - the client first checks whether there is an adb server process already running.
  - If there isn't, it starts the server process.
  - When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients.

Basics: ADB client commands

- If "*adb devices*" did not work as above you may need to kill server to re-run adb:
  - *sudo adb kill-server*
- Assuming everything is OK run adb to get to interactive Linux command-line shell on your device
  - *adb shell*
- The output to terminal should be like this (shell prompt in device)
  - *ginkgo:/$*
- Now at the device shell prompt you can use a number of Linux commands in regular way, e.g.
  - *"ls",*
  - *"ls /storage/self/primary",*
  - *"cat /proc/cpuinfo"*

Basics: ADB useful device shell commands

- Get frequencies of CPU cores:
  - *ginkgo:/$ cat /sys/devices/system/cpu/cpu{0,1,2,3,4,5,6,7}/cpufreq/scaling_available_frequencies*
- Set frequency of CPU core (300000 is taken from above output):
  - *ginkgo:/$ echo 300000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq*
- Get DDR frequencies:
  - *ginkgo:/$ cat /sys/class/devfreq/ddrfreq/available_frequencies*
- Investigate content of /sys/ using “ls” and TAB
  - *ginkgo:/$ ls /sys/*

Basics: uninstall bloatware from device

- List installed packages
  - *ginkgo:/$ pm list packages* | *grep facebook (запрещенная в России экстремистская организация)*
  - *ginkgo:/$ pm list packages* | *grep google*

- Decide what package to uninstall
  - *any facebook package is safe to uninstall (unless you use it)*
  - *search google/yandex for unneeded packages on your phone model*

- Uninstall whatever you decide to (google duo, google youtube, etc)
  - *ginkgo:/$ pm uninstall --user 0 com.google.android.youtube*

- Any google app can be installed again from Google Play Market
  - *do not uninstall play market and play services*

Basics: C/C++ console apps for device

- On your Linux machine install gcc/g++ cross-compiler for AARCH64 (Debian/Ubuntu)
  - *sudo apt install g++-aarch64-linux-gnu*
  - *need disk space for it on your Linux machine*
- Use your favorite editor to write “hell or paradise” program
  - *#include <stdio.h>*
  - *void main(void)*
  - *{*
    - *printf(“hell or paradise?\n”)*
  - *}*
- Cross-compile the program
  - *aarch64-linux-gnu-gcc –o hell.x hell.c*
- Push the executable hell.x to the device using adb (from Linux command prompt)
  - *adb push hell.x /data/local/tmp/*
  - *the folder /data/local/tmp/ is for executable files*

Basics: run C/C++ console app on device

- Run adb to get to interactive Linux command-line shell on your device
  - *adb shell*
- Recall: the output to Linux terminal should be like this (shell prompt in device)
  - *ginkgo:/$*
- Run hell.x on the device
  - */data/local/tmp/hell.x*
- Further development depends on success or failure of the previous step
  - *Not all device (OS) manufacturers allow to run command line applications on adb shell*

PMU and Linux perf: introduction

- PMU is (a HW) Performance Monitor Unit that provides performance information for kernel, processes, application, etc.
- Linux perf is a lightweight command-line utility for profiling and monitoring CPU performance on Linux systems
  - Must know for OS kernel optimizations, development, etc.
- Linux perf (Debian/Ubuntu) installation:
  - *sudo apt install linux-perf*
- Allow regular users to use perf:
  - *sudo su –*
  - *echo 0 > /proc/sys/kernel/perf_event_paranoid*
  - *exit*
- Revert the setting back (when not using perf):
  - *sudo su –*
  - *echo 3 > /proc/sys/kernel/perf_event_paranoid*
  - *exit*

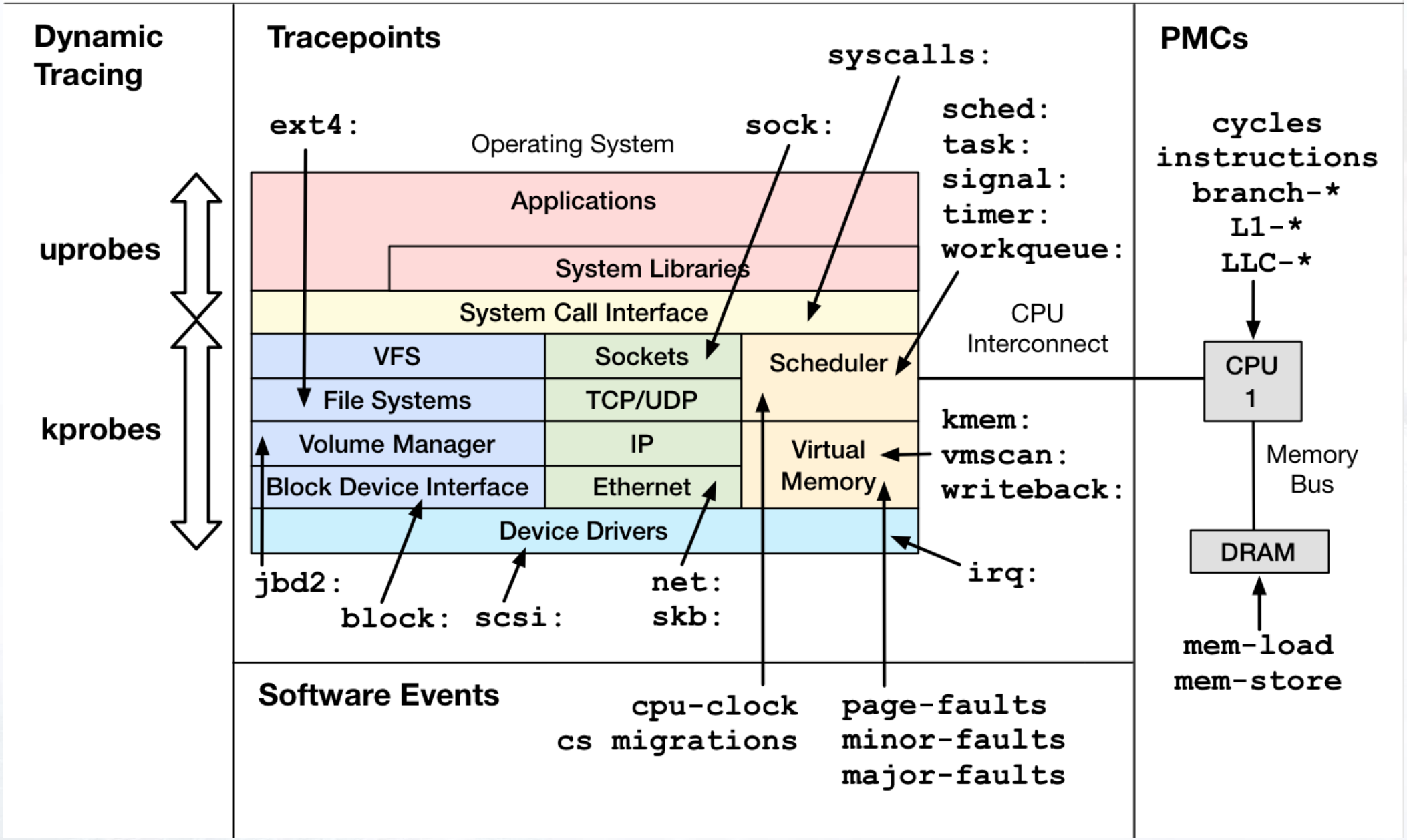
Hardware performance counters

- Hardware performance counters (HPC) or hardware counters are a set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems.
- Advanced SW (kernel scheduling) often rely on those counters to conduct low-level performance analysis or tuning
- Hardware counters provide low-overhead access to a wealth of detailed performance information related to CPU's functional units, caches and main memory etc.
- Another benefit of using them is that no source code modifications are needed in general.
- However, the types and meanings of hardware counters vary from one kind of architecture to another due to the variation in hardware organizations.

Using Linux perf on x86 for stats

- Compile the program for x86
  - *gcc –o hell.x hell.c*
- Run it under perf to collect statistics
  - *perf stat –e cycles –e instructions ./hell.x*
- Surprise – for a simple one liner app CPU spends:
  - *> 800,000 cycles*
  - *> 600,000 instructions*
- List all events for statistics on your platform:
  - *perf list*
- Use the perf stat events to evaluate your algorithm
  - Check perf Wiki at https://perf.wiki.kernel.org/index.php/Tutorial for more commands

Linux perf_events Event Sources



Collect PMU stats inside application

- Use system call perf_event_open
  - man perf_event_open

- The perf_event_open system call can be used to obtain performance information from inside a Linux/Android application.

- This system call does not have a glibc wrapper so it is called directly using syscall

- For convenience create a wrapper function, like the one shown in the manpage to make for easier usage.

Why Linux perf and perf_event_open?

- These may be used to collect stats, measure performance, find bottlenecks in your algorithms, your apps or in your OS

- Welcome with ideas and activities for premium exam