

B4B35OSY: Operační systémy

Android

Michal Sojka¹



18. prosince 2020

¹michal.sojka@cvut.cz

Obsah I

1 Úvod

2 Komponenty OS a Android

- Aplikace
- Souborový systém
- Init proces
 - SysV init
 - systemd
 - Android init
- Meziprocesní komunikace (IPC)
- Aplikace a frameworky (Android)

3 Závěr

Mobilní OS

- Dřívější OS byly jednodušší než desktopové OS
 - Symbian OS (Nokia), Windows CE, ...
- Dnes jsou mobily výkonné jako notebooky před pár lety
 - Mobilní OS jsou upravené verze desktopových
 - Android, iOS, (Windows Mobile)
- Tato přednáška bude převážně o OS Android, což je
 - Mobilní OS od Googlu (částečně open source)
 - Linuxové jádro (trochu změněné)
 - Jiný user space než mají běžné Linuxové distribuce (Ubuntu, Fedora, ...)

Čím se mobilní OS liší od „normálních“ OS?

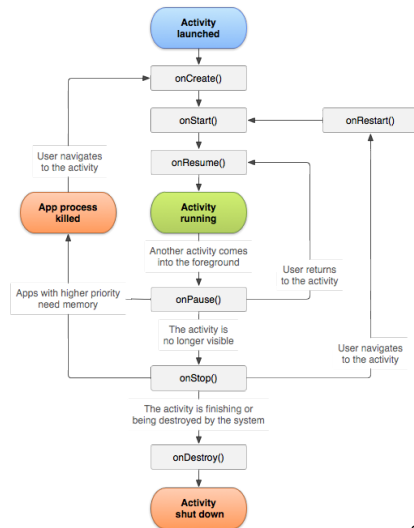
- Vše, co jsme probírali v předchozích přednáškách platí i pro mobilní OS:
 - Procesy/vlákná, synchronizace, správa paměti, IPC, ovladače, souborové systémy, grafika, ...
- To, co dělá mobilní OS mobilním jsou komponenty/knihovny/frameworky na vyšších úrovních
- Většina lidí pod pojmem „mobilní aplikace“ rozumí pouze tuto vyšší úroveň (tj. UI, design, ...)
- V této přednášce si zkusíme ukázat, jak některé vysokoúrovňové koncepty mobilních aplikací souvisí s nízkoúrovňovými záležitostmi probíranými dříve
- Podíváme se na některé komponenty či koncepty a ukážeme si, jak se liší od podobných komponent/konceptů v desktopových/serverových OS
 - Android vs. Linux na desktopu/serveru
 - Android se velmi rychle mění – ne vše, co je v této přednášce platí přesně pro poslední verze a/nebo všechny výrobce

Aplikace

- Aplikace se skládá z
 - kódu
 - zdrojů (resources) – obrázky apod.
 - manifest – popis aplikace
 - ...
- Kód
 - Většinou vyšší programovací jazyk (Java, Kotlin)
 - Může obsahovat i nativní kód (např. C/C++) volaný skrze Java Native Interface (JNI)
- App manifest
 - Jméno aplikace + ikona + popis
 - Seznam aktivit (+ intent filters), služeb, atd. a jejich implementaci (tříd)
 - Oprávnění, která aplikace potřebuje
 - Požadavky na HW a SW (např. minimální verze Androidu)

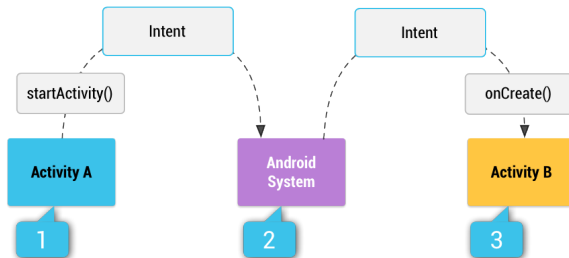
Aktivita

- Třída reprezentující jednu obrazovku
- Nejedná se o „imperativní“ kód, který běží od začátku do konce, ale o komponenty, které jsou volány (callback) různými frameworky (nižšími vrstvami).
- Programátor nemá kontrolu nad tím, kdy bude proces aplikace spuštěn a ukončen
 - Např. když je málo paměti
- Aktivita se spouští/přepínají tzv. **intenty**, což je forma meziprocesní komunikace



Intent

- Žádost o provedení akce v jiné komponentě
 - Např. spuštění aktivity
- Explicitní – je přesně řečeno, která aplikace akci provede
- Implicitní – uživatel si může vybrat aplikaci (např. otevření webové stránky)



Linux/UNIX

Filesystem Hierarchy Standard (FHS)

- Specifikuje hierarchii a obsah adresářů Linuxových distribucí
- Jedna aplikace je „rozprostřena“ do mnoha různých adresářů (s výjimkou /opt)
 - **/bin** – programy, UNIXové příkazy (dnes často symlink do /usr/bin)
 - **/boot** – soubory potřebné pro boot systému (jádro, initramfs)
 - **/dev** – pseudosoubory pro komunikaci s ovladači
 - **/etc** – konfigurace systému a jednotlivých programů
 - **/home** – domovské adresáře uživatelů
 - **/lib** – knihovny
 - **/media** – přípojná místa pro externí média (USB flash, CDROM, ...)
 - **/mnt** – dočasně připojené souborové systémy (např. síťové)
 - **/opt** – adresáře pro aplikační software – co adresář, to aplikace
 - **/proc** – virtuální souborový systém a informacemi o procesech a jádru
 - **/root** – domovský adresář správce systému
 - **/run** – RAM disk pro běhová data (zmizí po vypnutí systému)
 - **/sbin** – programy pro správce systému
 - **/srv** – data poskytovaná daným systémem (např. webovými servery)
 - **/sys** – virtuální souborový systém s informacemi o zařízeních apod.
 - **/tmp** – adresář pro dočasné soubory
 - **/usr** – podobná hierarchie jako v „/“ pro read-only data
 - **/var** – proměnné soubory (logy, mailboxy, data programů /var/lib, ...)

Android

Partitions

- **boot** – obsahuje jádro a počáteční RAM disk (initramfs)
- **cache** – cache pro stáhování aktualizací systému
- **recovery** – jádro a jiný initramfs pro obnovu systému
- **system** – /system
- **userdata** – /data

Souborový systém

- Je tvořen initramfs do kterého jsou připojeny (mount) adresáře z flash
 - **/init** – init proces (viz dále)
 - **/sbin** – kritické programy jako např. *adbd*, *healthd* a *recovery*
- Připojené adresáře
 - striktní oddělení systému a dat
 - **/system** – systémové komponenty Androidu – read-only
 - **/data** – uživatelská data
 - nejsou přemazána když se aktualizuje systém
 - lze je šifrovat

Android – hierarchie, obsah

/system

- **/app** – systémové aplikace (od Googlu či výrobce zařízení)
- **/bin** – nativní programy (dalvikvm, vold, ...), ladicí nástroje (adb, ...), UNIXové příkazy (cp, ls, ...), atd. ls, ...)
- **/etc** – konfigurace
- **/fonts**
- **/framework** – Javová část Androidy (.jar, .odex)
- **/lib** – nativní knihovny
- **/media** – zvuky, animace, ...
- **/prov-app** – Privileged Application
- **/usr** – Support file (keyboard mappings, ...)
- **/vendor**
- **/xbin** – další systémové programy, většinou pro ladění (strace, tcpdump, nc, ...)

/data

- **/app** – balíky .apk instalovaných aplikací
- **/backup**
- **/dalvik-cache**
- **/data** – aplikace si tam mohou uchovávat svá data (viz níže)
- **/media** – připojená SD karta
- **/misc** – konfigurace, klíče, ...
- **/property** – uložené „vlastnosti“, které přežijí reboot
- **/user** – pro podporu více uživatelů
- **/system**
- ...

/data/data

- **/com.android.providers.calendar** – obsahuje databases/calendar.db
- **/com.android.providers.contacts** – obsahuje databases/contacts2.db
- **/com.android.chrome** ...
- ...

Uživatelé

- Android používá systémové „uživatele“ (UID) jinak, než běžné Linuxové distribuce
- Každá aplikace běží s právy jiného uživatele (UID tedy identifikuje aplikaci, ne uživatele)
- Tím je (mimo jiné) zajištěna ochrana dat jedné aplikace před ostatními
- Některá zařízení/verze Androidu podporují více uživatelů (lidí) – každému uživateli je přiřazeno 100000 UID (může tedy nainstalovat 100000 aplikací).

Init proces

- V UNIXových OS je `init` první proces, který je spuštěn po zavedení jádra OS
 - Něco jako `user/hello` v naší verzi OS NOVA
- Jeho úkolem je:
 - Připojit potřebné souborové systémy
 - Spustit severy a daemony potřebné pro běh systémů
 - Spustit proces(y), které umožní lokální přihlášení uživatele (`getty` pro textovou konzoli, *display manager* pro grafické přihlášení)
 - Adoptovat procesy, jimž umře rodič

SysV init

- *UNIX System V (system five)* je jedna z verzí komerčního UNIXu od AT&T (1983)
 - Init proces je vytvořen spuštěním `/sbin/init`
 - Načte `/etc/inittab` a vykoná, co je tam napsáno
-
- Runlevel = co se má spustit při bootování vypínání
 - Vždy se spustí skript `rcS` – základní inicializace a služby systému
 - Poté se spustí skript `rc <N>2`, který spustí další služby (webový server, grafický login, ...)
 - Getty (textový login) se spustí v runlevelech 2–5 a při ukončení se spustí znovu

Example (/etc/inittab:)

```
id:2:initdefault:
si::sysinit:/etc/init.d/rcS

# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.
10:0:wait:/etc/init.d/rc 0
# ...
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6

1:2345:respawn:/sbin/getty 38400 tty1
```

²rc je zkratka „runcom“ (run commands) – viz <https://kb.iu.edu/d/abnd>

SysV init – pokračování

- `respawn` v *inittab* aktivuje „monitorování procesu“ a restartuje proces např. v případě nečekaného pádu
- SysV init se mnohdy často používá v případě jednoduchých „embedded“ zařízení, kdy v systému běží jen pár služeb

Skripty `rc` a `rcS`

- Spouští ostatní služby na základě tzv. *init skriptů* (někdy také *rc skriptů*)
- Jednoduchá implementace `rcS` sekvenčně spouští skripty z adresáře `/etc/rcS` začínající na „S“ (start) nebo „K“ (kill):

```
for i in /etc/rcS.d/S??*; do $i start; done
```
- Příklad jmen `init` skriptů: `S01hostname`, `S02udev`, `S15networking`
- Často to jsou pouze symbolické odkazy na skripty v adresáři `/etc/init.d`
 - `/etc/rcS.d/S15networking` -> `/etc/init.d/networking`
 - `/etc/rc0.d/K08networking` -> `/etc/init.d/networking`

Problémy

- Řešení závislostí mezi službami pořadím startování
- Paralelní spouštění služeb (multi-core CPU)
- Monitorování a restart havarovaných služeb
- ... vše se dá řešit pomocí různých „nastaveb“, ale ...

Příklad jednoduchého init skriptu

Example (S40network)

```
#!/bin/sh

mkdir -p /run/network

case "$1" in
    start)
        printf "Starting network: "
        /sbin/ifup -a
        [ $? = 0 ] && echo "OK" || echo "FAIL"
        ;;
    stop)
        printf "Stopping network: "
        /sbin/ifdown -a
        [ $? = 0 ] && echo "OK" || echo "FAIL"
        ;;
    restart|reload)
        "$0" stop
        "$0" start
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac

exit $?
```

systemd

- Moderní init systém pro Linux
- Řeší většinu problémů SysV init (a přináší jiné problémy)
- Umožňuje popsání závislostí mezi službami => paralelní spouštění
- Aktivace pomocí socketů – viz „Socket activation“ dále
- Snadné nastavení zabezpečení služeb a přidělování zdrojů
 - Např. omezení množství paměti a CPU pro danou službu
 - Toto implementuje linuxové jádro; `systemd` umožňuje pouze snadnou konfiguraci

Watchdog

- Možnost periodické komunikace se službou
- Pokud se služba dlouho nehlásí, `systemd` ji restartuje

Popis služby pro systemd (.service file)

[Unit]

Description=Deferred execution scheduler

Documentation=man:atd(8)

After=remote-fs.target nss-user-lookup.target

[Service]

ExecStart=/usr/sbin/atd -f

IgnoreSIGPIPE=false

KillMode=process

Restart=on-failure

[Install]

WantedBy=multi-user.target

Socket activation

■ Co to je?

- Elegantní řešení závislostí bez nutnosti jejich explicitního popisu
- Služby jsou startovány jen/až když je někdo potřebuje

■ Závislosti

- Servery často poskytují své služby pomocí socketů (UNIX, TCP/localhost, ...)
- Pokud služba (proces) A potřebuje něco od služby B, připojí se k socketu služby B a pošle požadavek.
 - Může se stát, že B také potřebuje něco od A. Kterou službu spustit jako první?

■ Základní myšlenka:

- 1 Vytvořit sockety všech služeb (ale ne jejich procesy)
- 2 Pokud se někdo k socketu připojí, spustit proces a předat mu už „existující“ socket

■ Implementace:

- Služba nevytváří socket sama, ale nechá to na *systemd* (konfigurační soubor).
- Při spuštění služba „zdědí“ socket od *systemd* (`fork()`, `exec()`), který jí sdělí, který file descriptor odpovídá socketu
- Služba tedy nedostane od „systému“ jen `stdin`, `stdout` a `stderr`, ale i socket, kterým klienti posílají požadavky

Android init

- Vzdáleně podobný SysV init
- Obsahuje navíc „System Properties“
- Místo `/etc/inittab` má `/init.rc`, `/init.usb.rc`/ apod.
- Nemá „runlevely“, ale umí spouštět služby na základě „triggers“ a „system properties“
 - např. při změně property se spustí/restartuje služba – podobné jako `launchd` v iOS)
 - Příklad změny property: Připojení k nabíječce, připojení USB, ...
- Služby jsou automaticky restartovány, pokud nejsou nakonfigurovány jako `oneshot`.
- Pokud je služba označena jako `critical` a nejde restartovat, je restartováno celé zařízení
- Podpora socket activation pro UNIX sockety
- Specifické *rc skripty*
- S jinou konfigurací funguje jako `ueventd` (další služba v OS Android)

System properties

- Jsou uloženy v několika souborech (dané výrobcem `/system/default.prop`, persistentní `/data/property/persist*`, ...)
- Přístup k properties přes `/dev/socket/property_service`, kontrola přístupu podle UID, možnost mapování do paměti (`mmap`).
- Příklady „properties“:
 - `wlan.driver.status`, `net.hostname`, `sys.boot_completed`, `net.dns1`, ...

.rc soubory

Example (init.rc – zkráceno)

```
on boot
    ifup lo
    hostname localhost
    domainname localdomain
    write /proc/sys/net/core/xfrm_acq_expires 3600
service ueventd /system/bin/ueventd
    class core
    critical
    seclabel u:r:ueventd:s0
    shutdown critical
service console /system/bin/sh
    class core
    console
    disabled
    user shell
    group shell log readproc
    seclabel u:r:shell:s0
    setenv HOSTNAME console
on property:ro.debuggable=1
    # Give writes to anyone for the trace folder on debug builds.
    # The folder is used to store method traces.
    chmod 0773 /data/misc/trace
    # Give reads to anyone for the window trace folder on debug builds.
    chmod 0775 /data/misc/wmtrace
    start console
```

Zygote

- Jedním z procesů spouštěných procesem `init` je tzv. `zygote` (uložen v `/system/bin/app_process`)
- Urychluje spouštění aplikací
- Spustí Dalvik Virtual Machine a načte všechny frameworky (třídy) OS android
- Zastaví se těsně před „načtením“ hlavní třídy aplikace, otevře `/dev/socket/zygote` a čeká na požadavky
- Přicházející požadavky obsahují jméno třídy aplikace
 - Zygote zavolá `fork()` a načte třídu aplikace
 - Fork používá mechanismus **copy-on-write**
 - Tímto způsobem se velmi rychle vytvoří proces aplikace, protože vše (JVM, frameworky, ...) už je nainicializované

Nízkoúrovňová IPC

Obecně (v UNIXu/Linuxu)

- roura (pipe)
 - přenášení proudu dat mezi dvěma procesy jedním směrem
- UNIX socket
 - přenášení dat (proud nebo zprávy) mezi dvěma procesy (obousměrné)
 - může, ale nemusí být vidět v souborovém systému (např. `/run/cups/cups.sock` pro komunikaci s tiskovým serverem CUPS)
 - lze přenášet i „file descriptor“ (FD) mezi různými procesy
 - Příklad: Privilegovaný proces otevře soubor a pošle FD jinému procesu, který soubor sám otevřít nemůže.

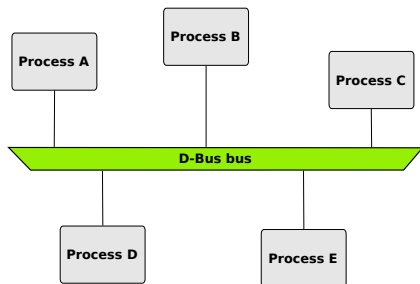
Remote Procedure Call (RPC)

Obecně

- Možnost volat funkce/procedury ve vzdáleném procesu
- Princip:
 - 1 Při zavolání funkce se provede serializace parametrů (převod dat v paměti do formátu pro komunikaci) a odešle se žádost (data) cílovému procesu (např. pomocí socketu).
 - 2 Cílový proces data deserializuje, zjistí jakou funkci má zavolat a zavolá ji
 - 3 Pokud funkce něco vrací, výsledek se serializuje a odešle zpět.

DBus (Desktop Linux)

- Mnoho aplikací potřebuje komunikovat na vyšší úrovni než posílání zpráv
 - Publish/subscribe
 - Komunikace jednotlivých objektů/komponent uvnitř aplikací
 - Nechce řešit, který socket použít pro danou aplikaci (v jakém procesu se nachází atd.)
 - ...
- DBus je systémový démon, který umožňuje aplikacím komunikovat na vyšší úrovni
- Aplikace si mohou definovat objekty, ptát se na objekty v jiných aplikacích, žádat o notifikace na změny v jiných aplikacích apod.



© 2015 Javier Cantero - this work is under the Creative Commons Attribution ShareAlike 4.0 license

Android Binder

- Poskytuje RPC
- Narozdíl o DBus je implementován v jádře
- `/dev/binder`
- Služby:
 - 1 Hledání cílového procesu
 - 2 Přenos zpráv
 - Android Interface Definition Language (AIDL) – generuje kód, který převádí volání funkcí na komunikaci pomocí Binderu (serializace/deserializace)
 - Blokuje (`ioctl(BINDER_WRITE_READ)`)
 - 3 Přenos objektů
 - file descriptor
 - 4 Důvěryhodné ověření zdroje
 - adresát ví, kdo mu zprávu poslal (PID, UID)

Dalvik VM

- Implementace Java VM od Googlu
- Aplikace se kompilují „just-in-time“ (JIT) překladačem do nativního kódu (výsledky se cachují)
- Dalvik má různé problémy – novější verze přecházejí na ART, kde se používá Ahead-of-time (AOT) kompilace

system_server

- Proces, kde různé systémové služby běží jako vlákna
- Podobný `svchost.exe` z Windows (služby jsou nahrávány z DLL knihoven)
- Psaný v Javě, služby jsou třídy v Javě
- Po inicializaci je spuštěna hlavní smyčka, která čeká na požadavky z jiných procesů a předává je službám
- Poskytované služby:
 - 1 **Bootstrap**: Installer, ActivityManager, PowerManager, DisplayManager, PackageManager, UserManager
 - 2 **Základní (Core)**: Lights, Battery, UsageStats, WebViewUpdate
 - 3 **Ostatní**: ...

Reference

- Jonathan Levin, *Android Internals: A Confectioner's Cookbook*, Technologeeks.com, 2015, <http://newandroidbook.com/>