Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Úpravy operačního systému Android

Martin Pozděna

Vedoucí práce: Ing. Josef Gattermayer

2. ledna 2014

Poděkování

Rád bych touto cestou poděkoval Ing. Josefu Gattermayerovi za námět zajímavého tématu bakalářské práce, zapůjčení nezbytného hardwaru a příkladné vedení celé práce. Dál bych rád poděkoval všem, kteří po mně bakalářskou práci přečetli a dopomohli s korekturou textu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 2. ledna 2014 .....

Odkaz na tuto práci

Pozděna, Martin. *Úpravy operačního systému Android*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

This thesis contains description of changes which can be made to OS Android in Ainol Novo 7 Crystal tablet in order to avoid turning off ethernet interfaces when display is turned off. Process of downloading and building OS Android source codes as well as OS Android architecture is covered throughout this work. Moreover, this thesis contains analysis of possibilities of OS Android modifications and remote update over SSH in Ainol Novo 7 Crystal tablet.

Keywords Android, Operating system, Build, Ethernet Manager, Wakelock, Remote update, Modifications, Tablet, Ainol

Abstrakt

Práce obsahuje popis úprav OS Android v tabletu Ainol Novo 7 Crystal, které odstraní vypínaní ethernet rozhraní při vypnutí displeje. Je zde popsán proces získání zdrojových kódů, jejich sestavení a architektura OS Android. Práce dále obsahuje rozbor možností úprav OS Android v tabletu Ainol Novo 7 Crystal a postup vzdálené aktualizace OS Android v tomto tabletu pomocí SSH.

Klíčová slova Android, Operační systém, Sestavení, Ethernet Manager, Wakelock, Vzdálená aktualizace, Úpravy, Tablet, Ainol

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Operační systém Android se od svého uvedení na trh v roce 2008 těší stále větší popularitě. První verze OS Android byly určeny pouze pro nasazení v mobilním telefonu, ale s rostoucí popularitou se OS Android postupně rozšířil i do jiných typů zařízení, především tabletů. Takové zařízení může plnit i funkce, kde by tradičně byla nasazena nějaká forma vestavného systému.

Tato práce vychází z použití tabletu s OS Android v systému domovního vrátného. V každé bytové jednotce domu je instalován tablet Ainol Novo 7 Crystal, který je zapojen do místní sítě pomocí USB Ethernet adaptéru. Tablet je prostřednictvím této místní sítě propojen s libovolným počtem bezpečnostních kamer v domě a libovolným počtem zvonkových systémů u vchodů.

V současnosti již existuje použitelné řešení výše popsaného systému, které však má technický nedostatek. Z hlediska hardwaru je již vše funkční. Veškeré komponenty (tablet, kamery a zvonky) spolu mohou bez problémů komunikovat po síti. Tablet je v bytě nainstalován do elegantního kovového pouzdra přišroubovaného na zeď a napájen z místní sítě na základě technologie Power over Ethernet. V tabletu jsou nainstalovány aplikace, které zajišťují všechny funkcionality očekávané od systému domovního vrátného.

Sestavení OS Android, které beží v tabletu, má jednu nevyhovující vlastnost. Základním požadavkem na tablet je, že musí být nepřetržitě dostupný na místní síti. Z tohoto důvodu jedna z aplikací drží neustále zámek PARTIAL_WAKE_LOCK, který má zabránit v uspání tabletu za účelem úspory energie. Tablet však při každém vypnutí displeje přestane komunikovat po síti, čímž přestane plnit svou funkci, neboť se žádný z návštěvníků nedovolá do bytové jednotky.

Předpokládá se, že se jedná o důsledek špatně implementovaného zámku PARTIAL_WAKE_LOCK společností Ainol. Tabletu by v takovém pří-

padě nic nebránilo při každém vypnutí displeje usnout, což by mělo za následek popisovanou závadu.

Hlavním cílem této práce je odstranění výše popsaného nedostatku, který omezuje fungování celého systému. Na začátku práce nebyl znám žádný postup, kterým by bylo možné takové úpravy dosáhnout. Je tedy nutné detailně analyzovat popsanou závadu, najít možnosti jejího odstranění, odstranit ji a vytvořit aktualizaci OS, která tyto změny promítne do tabletu.

Vzhledem k faktu, že zdrojové kódy OS Android jsou volně dostupné, je prvním přirozeným krokem zjištění, jakým způsobem lze tyto zdrojové kódy získat. Dále je potřeba zjistit, jak tyto zdrojové kódy sestavit ve fungující OS a je-li reálné získat takovou variantu zdrojových kódů, kterou bude možné použít v tabletu Ainol Novo 7 Crystal. Dalším krokem, který je třeba vykonat, je zjištění možností úprav operačního systému Android v tomto konkrétním tabletu.

Tablety jsou instalovány v obydlených bytových jednotkách, do kterých je možné vstupovat pouze ve výjimečných případech se souhlasem majitele. Z tohoto důvodu není vhodné provádět aktualizaci OS Android pomocí USB připojení k počítači, jak je u tohoto tabletu běžné. Je tedy nutné zjistit možnosti vzdálené aktualizace OS přes protokol SSH a navrhnout postup takové aktualizace.

Výstupem této práce by měla být aktualizace, která zajistí korektní fungování ethernet rozhraní tabletu při vypnutém displeji. Tuto aktualizaci musí být následně možné nainstalovat do tabletu vzdáleně pomocí protokolu SSH.

Kapitola 1

Android Open Source Project

1.1 Úvod do AOSP

AOSP nebo-li Android Open Source Project tvoří spolu s upraveným jádrem Linuxu, na kterém běží, operační systém Android.[29] Android je operační systém, který je určen především pro použití v mobilních zařízeních, jako jsou mobilní telefony nebo tablety, ale začíná se také objevovat v digitálních fotoaparátech, televizích nebo vestavných systémech.

Historie Androidu sahá na konec roku 2003, kdy Andy Rubin, Rich Miner, Nick Sears a Chris White založili v Palo Alto, USA firmu Android Incorporation, která se zabývala vývojem softwaru pro mobilní zařízení. Google svůj zájem o tuto oblast projevil v srpnu 2005, když koupil celou firmu Android Incorporated. Dva roky poté, v listopadu 2007, byla nejdříve ustavena aliance firem Open Handset Alliance, jejichž cílem bylo vytvoření otevřeného standardu pro mobilní zařízení. Následně byla vydána první betaverze Android SDK, nástrojů, které umožňují tvorbu aplikací pro platformu Android. Prvním mobilním telefonem využívajícím operační systém Android verze 1.0 byl 23. září 2008 HTC Dream G1.[28]

Mobilní zařízení s open source operačním systémem Android se od té doby těší stále větší popularitě, která bude podle předpokladů dále narůstat. K prosinci 2013 je Android nejpoužívanějším operačním systémem pro mobilní zařízení s více než 1 000 000 nových zařízení aktivovaných každý den. Google Play, které je největším, ale ne jediným distributorem aplikací pro tuto platformu, zaznamená přes 1 500 000 000 stažených aplikací každý měsíc.[11]

Zdrojové kódy AOSP jsou volně přístupné na internetu, ale jeho vývojový cyklus je od většiny open source projektů odlišný. Jeho vývoj se odehrává v Googlu za zavřenými dveřmi a veřejnost nemá přístup k diskuzi

o směřování vývoje ani k aktuálním zdrojovým kódům. Při vývoji nové verze Androidu Google obvykle spolupracuje s některým výrobcem hardwaru (Samsung, HTC, LG, Asus, ...). Když je nová verze hotová, představí ve spolupráci s výrobcem zařízení s touto verzí a zveřejní její zdrojové kódy. Nevýhodou tohoto přístupu je, že se o změnách v AOSP dozvíte pouze analyzováním zdrojových kódů, které jsou málo okomentované a dokumentace k nim prakticky neexistuje. Výhodou naopak je možnost Googlu držet veřejnost v očekávání, jaké funkcionality nová verze přinese a vzbudit větší zájem o Android.[29]

Jedinou odchylkou od obvyklého vývojového cyklu byla verze 3.x, která je určena pouze pro tablety. Google chtěl nejspíš předejít fragmentaci Android platformy a zdrojové kódy k této verzi nikdy nezveřejnil.[3][6] Ve verzi 4.0 se obě vývojové větve spojily v jednu.[29]

V tabulce 1.1 jsou zobrazeny jednotlivé verze Androidu pro všechny doposud vydané API úrovně. API je v tomto kontextu myšleno rozhraní, které mohou aplikace využívat k interakci s operačním systémem a jeho úroveň je varianta, která je v dané verzi Androidu k dispozici.[16] Verzi Androidu je možné zjistit v `Nastavení -> Informace o tabletu` nebo terminálovým příkazem `getprop ro.build.version.release`. Tablet Ainol Novo 7 Crystal obsahuje OS Android ve verzi 4.1.1. Pro každou verzi Androidu existuje obvykle více variant zdrojových kódů. V případě Androidu 4.1.1 je to celkem 8 variant.[3]

1.2 Získání zdrojových kódů

K sestavení fungujího operačního systému Android je třeba získat zdrojové kódy dvou součástí: Android-kompatibilního jádra Linuxu a Android platformy.[29] Android platforma je hlavní součástí Android Open Source Projectu a nadále budou oba názvy používány jako synonyma popisující všechny součásti OS Android kromě jádra.

1.2.1 Android-kompatibilní jádro Linuxu

Samotná Android platforma běží nad jádrem Linuxu, které ale není jeho součástí. Google během vývoje Androidu přidal do standardního jádra Linuxu, které je k nalezení na `https://www.kernel.org/`, několik mechanismů. Jedná se například o wakelock, binder, ashmem (Anonymous Shared Memory), alarm a logger, které optimalizují jádro pro běh na mobilním zařízení, ale zároveň nebyla zachována kompatibilita s originálním jádrem.[29] Z tohoto důvodu je potřeba použít Android-kompatibilní linuxové jádro.

Tabulka 1.1: Verze Androidu[3][29]

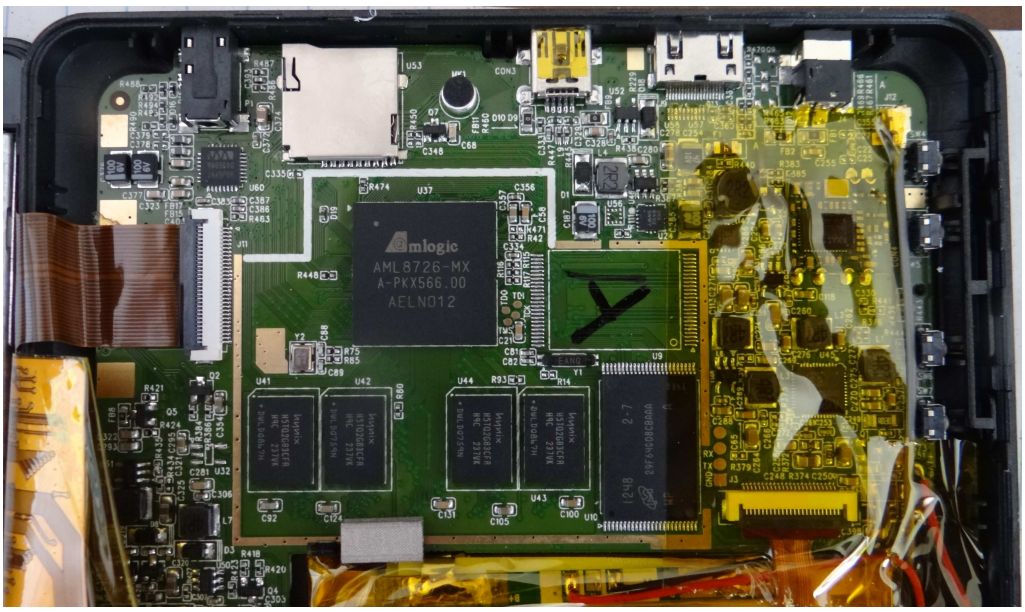
Verze	Vydáno	Kódové označení	úroveň API
1.0	září 2008		1
1.1	únor 2009		2
1.5	duben 2009	Cupcake	3
1.6	září 2009	Donut	4
2.0	říjen 2009	Eclair	5
2.0.1	prosinec 2009	Eclair	6
2.1	leden 2010	Eclair	7
2.2.x	květen 2010	Froyo	8
2.3–2.3.2	prosinec 2010	Gingerbread	9
2.3.3–2.3.7	únor 2011	Gingerbread	10
3.0	leden 2011	Honeycomb	11
3.1	květen 2011	Honeycomb	12
3.2.x	červenec 2011	Honeycomb	13
4.0.1–4.0.2	listopad 2011	Ice-Cream Sandwich	14
4.0.3–4.0.4	prosinec 2011	Ice-Cream Sandwich	15
4.1.x	červen 2012	Jelly Bean	16
4.2.x	listopad 2012	Jelly Bean	17
4.3.x	červenec 2013	Jelly Bean	18
4.4	říjen 2013	KitKat	19

Výjimku tvoří hrstka zařízení, jejichž jádro je z různých důvodů součástí AOSP. I v tomto případě je nutné jádro kompilovat odděleně od zbytku AOSP.[1] Na žádné Ainol zařízení se tato výjimka nevztahuje.

Jednou z možností, jak získat Android-kompatibilní jádro, je provést patch originálního jádra.[7]

Druhou možností je získat jádro přímo od výrobce konkrétního System on a Chip (dále SoC), který je použit v zařízení na které je cíleno sestavení Androidu. V tomto případě nebude potřeba řešit žádné problémy s ovladači, protože jádro už bylo upraveno pro konkrétní hardware výrobcem. Vzhledem ke stoupající popularitě Androidu se většina výrobců snaží nabízet zdrojové kódy, buď jádra nebo celého Androidu, upravené pro jejich hardware.[29]

Jak je vidět na obrázku 1.1, je v tabletu Ainol Novo 7 Crystal použit SoC Amlogic AML8726-MX. Amlogic nabízí zdrojové kódy celého OS Android (včetně jádra) zde: `http://openlinux.amlogic.com/Android/`



Obrázek 1.1: SoC tabletu Ainol Novo 7 Crystal

`Tablet`. Tato možnost je však omezena pouze na zákazníky, kteří od Amlogic jejich SoC kupují. Zdrojové kódy se tedy touto cestou získat nepodařilo. Později se ukázalo, že ke změně požadovaných vlastností tabletu není třeba měnit vlastnosti jádra, takže jeho zdrojové kódy nejsou nezbytně nutné.

1.2.2 Android platforma

Všechny zdrojové kódy AOSP jsou zveřejněny na stránkách `https://android.googlesource.com/` ve formě git repozitářů. Jedná se o repozitáře jednotlivých součástí systému, které je možné stáhnout jednotlivě příkazem `git clone https://android.googlesource.com/name`, kde `name` je název repozitáře ke stažení.

Stahování všech repozitářů, které jsou zde zveřejněny by bylo neefektivní, protože pro kompletní Android platformu je potřeba pouze jejich podmnožina.[29] Ke korektnímu stažení kompletní Android platformy slouží nástroj `repo`, který lze získat následovně:[4]

```
$ curl http://commondatastorage.googleapis.com/\
> git-repo-downloads/repo > repo
```

Stáhnout zdrojové kódy Android platformy do aktuální pracovní složky je možné takto:[4]

```
$ repo init -u https://android.googlesource.com/platform/\
> manifest -b android-4.1.1_r1
$ repo sync -j16
```

První příkaz inicializuje `repo` v aktuální složce. Parametr `-u` udává cestu k XML souboru, který definuje jednotlivé verze Android platformy. Parametr `-b` udává verzi Android platformy, která má být stažena.[29] Verze Android platformy není to samé jako verze Androidu, jedna verze Androidu může mít více verzí platformy. Seznam všech verzí Android platformy je na `http://source.android.com/source/build-numbers.html`. Požadovaný parametr je k nalezení v tabulce pod názvem *Tag*. Pro potřeby bakalářské práce jsem pracoval s verzí platformy `android-4.1.1_r1`.

Druhý příkaz se postará o samotné stažení zdrojových kódů do aktuální složky. Parametr `-j` udává počet simultánně běžících stahování jednotlivých git repozitářů. Implicitní hodnota jsou čtyři simultánní stahování. V některých případech může být výhodné tuto hodnotu změnit. Zdrojové kódy Android platformy se ve verzi `android-4.1.1_r1` skládají z více než 300 000 souborů o celkové velikosti přes 5 GB. Na počítači s rychlostí připojení 100 Mbitů trvá stažení s implicitní hodnotou `-j4` 25 minut, v případě použití hodnoty `-j16` 23 minut.

1.2.3 Součásti AOSP

V tabulce 1.2 jsou popsány složky, které jsou součástí AOSP verze android-4.1.1_r1. Význam složky *gdk/* není znám[29] a pro potřeby této práce není důležitý. Detailnější popis některých významných částí AOSP a jejich vzájemné komunikace je blíže popsán v sekci 2.1. V kořenové složce AOSP se ještě nachází soubor *Makefile*, který obsahuje následující:

```
### DO NOT EDIT THIS FILE ###
include build/core/main.mk
### DO NOT EDIT THIS FILE ###
```

Soubor slouží k začlenění základního konfiguračního souboru do Android sestavovacího systému.[29]

1.3 Základní technika sestavení

1.3.1 Příprava prostředí pro sestavení

Android sestavovací systém je testován Googlem na OS Ubuntu LTS (10.04), ale Mac OS a většina ostatních linuxových distribucí by měla být

Tabulka 1.2: Součásti AOSP[29]

Složka	**Obsah**	**Velikost v MB**
abi	základní podpora C++	0.07
bionic	Android specifická C knihovna (Android náhrada knihovny libc)	9.3
bootable	recovery a referenční bootloader	3.9
build	build system	3.5
cts	compatibility test suite	86
dalvik	Dalvik virtual machine	27
development	vývojářské nástroje	71
device	device-specific soubory	31
docs	obsah `source.android.com`	4.9
external	externí projekty používané v AOSP	1 363
frameworks	klíčové komponenty AOSP především system services	933
gdk		5.6
hardware	HW knihovny	35
libcore	Apache Harmony	31
libnativehelper	pomocné funkce pro použití JNI	0.1
ndk	native development kit	24
packages	základní Android aplikace	270
pdk	platform development kit	0.02
prebuilt	předkompilované binárky	291
prebuilts	předkompilované binárky (postupně nahrazuje prebuilt)	1 855
sdk	software development kit	36
system	nativní aplikace a daemoni	6.3
celkem		**5 087**

schopná zkompilovat korektně celou Android platformu. Od verze Androidu 2.3.x je pro sestavení vyžadováno 64-bitové prostředí. Pro jedno sestavení je potřeba přibližně 30 GB volného místa na disku a následující nástroje:[5]

- Python 2.6–2.7
- GNU Make 3.81–3.82
- JDK 6 (pro verze AOSP 2.3.x a novější)

- Git 1.7 nebo novější

Instalaci požadovaných balíčků kromě JDK na Ubuntu 12.04 lze provést takto:[5]

```
$ sudo apt-get install git gnupg flex bison gperf \
> build-essential zip curl libc6-dev libncurses5-dev:i386 \
> x11proto-core-dev libx11-dev:i386 libreadline6-dev:i386 \
> libgl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib mingw32 \
> tofrodos python-markdown libxml2-utils xsltproc \
> zlib1g-dev:i386
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 \
> /usr/lib/i386-linux-gnu/libGL.so
```

AOSP vyžaduje pro správné sestavení JDK 6. JDK balíček od společnosti Oracle ovšem není nadále v repozitářích Ubuntu a není tedy možné jej snadno nainstalovat příkazem `apt-get`.[22] Je nutné instalaci provést ručně stažením nejaktuálnější verze JDK 6 ze stránek Oracle.[18] V okamžiku psaní tohoto textu se jedná o soubor *jdk-6u45-linux-x64.bin*. Instalace se provede následovně:[29]

```
$ chmod 755 jdk-6u45-linux-x64.bin
$ ./jdk-6u45-linux-x64.bin
$ sudo mkdir -p /usr/lib/jvm
$ sudo mv jdk1.6.0_45 /usr/lib/jvm/
$ sudo update-alternatives --install "/usr/bin/java" "java" \
> "/usr/lib/jvm/jdk1.6.0_45/bin/java" 1
$ sudo update-alternatives --install "/usr/bin/javac" \
> "javac" "/usr/lib/jvm/jdk1.6.0_45/bin/javac" 1
$ sudo update-alternatives --install "/usr/bin/javah" \
> "javah" "/usr/lib/jvm/jdk1.6.0_45/bin/javah" 1
$ sudo update-alternatives --install "/usr/bin/javadoc" \
> "javadoc" "/usr/lib/jvm/jdk1.6.0_45/bin/javadoc" 1
$ sudo update-alternatives --install "/usr/bin/jar" "jar" \
> "/usr/lib/jvm/jdk1.6.0_45/bin/jar" 1
```

Nyní je Sun JDK v počítači nainstalován. Je ještě třeba vybrat jej jako preferované JDK na úkor OpenJDK, který se v Ubuntu běžně používá, takto:[29]

```
$ sudo update-alternatives --config java
```

Ověřit správnost nastavení lze příkazem:[29]

```
$ sudo update-alternatives --display java
java - manual mode
link currently points to /usr/lib/jwm/jdk1.6.0_45/bin/java
...
```

Obdobně je třeba nastavit i součásti `javac`, `javah`, `javadoc` a `jar`.[29] Nyní je počítač připravený k sestavení Android platformy.

1.3.2 Sestavení

Pro sestavení musí být pracovní složka terminálu nastavena na kořenovou složku stažených zdrojových kódů AOSP. Dále budu všechny cesty k souborům uvádět jako relativní vzhledem k této složce. Prvním krokem je nastavení prostředí, zavoláním skriptu `build/envsetup.sh`. Je nežádoucí, aby skript běžel v novém shellu, protože by skript nastavil pouze nový shell. Je potřeba zavolat jej jako `source build/envsetup.sh` nebo `. build/envsetup.sh`.[2] K dalšímu nastavení parametrů kompilace slouží příkaz `lunch`, k jeho správnému fungování je nutné mít nastavené prostředí z prvního kroku.[29] Pomocí `lunch` je možné zvolit parametry sestavení jako je CPU architektura, cílový produkt a typ sestavení, které program interaktivně nabídne. Implicitní varianta `full-eng` je učena pro spuštění na emulátoru. Samotná kompilace je spuštěna zavoláním příkazu `make -j4`, kde přepínač `-j` udává počet simultánně běžících úloh. Na notebooku Intel Core i5 s 4 GB RAM a čtyřmi simultánně zpracovávanými úlohami trvá kompilace verze `android-4.1.1_r1` přibližně 2 hodiny.

Výstupem je obsah složky *out*, která v tomto případě obsahuje přibližně 16 GB dat. Při úspěšně dokončeném sestavení je možné systém spustit v emulátoru pomocí příkazu `emulator`.[5][29] Pro spuštění emulátoru je potřeba mít nastavené prostředí skripty `build/envsetup.sh` a `lunch`. Pokud byl po dokončení kompilace vypnut daný terminál, je třeba opětovně nastavit prostředí danými příkazy. Dříve jsem uvedl, že součástí AOSP není jádro, které je třeba získat odděleně. V případě emulátoru je zde výjimka, kdy ve složce *prebuilts/qemu-kernel/* je zkompilované jádro pro ARM i x86 architekturu.

V kapitole jsem se nezabýval samotnou kompilací jádra, protože jádro pro potřeby této práce nebylo třeba kompilovat. Samotné změny provedené v jádře Googlem příliš nemění postup jeho kompilace, který je uveden například v knize *Linux Kernel in a Nutshell*.[17]

1.4 Pokročilejší techniky sestavení

Android sestavovací systém se výrazně liší od systému, který je běžný například pro sestavování jádra Linuxu. V případě Linuxu existuje jeden hlavní makefile, který rekurzivně volá další makefily z podsložek. Naproti tomu u Androidu jsou procházeny podsložky až do nalezení prvního konfiguračního souboru *Android.mk* a pokud není v tomto souboru definováno jinak, hlubší podsložky se již neprochází. Každý konfigurační soubor *Android.mk* udává, jak zkompilovat jeden modul Android platformy.[29] Takovýchto modulů je v AOSP verze `android-4.1.1_r1` celkem 1 828. Po každém zavolání příkazu `make` je nejdříve zobrazena konfigurace sestavení a poté sestavovací systém projde všechny relevantní *Android.mk* soubory a vytvoří z nich jeden velký konfigurační soubor podle kterého celé AOSP zkompiluje.[29]

Další rozdílem oproti rekurzivnímu sestavovacímu systému je způsob konfigurace. Zatímco rekurzivní systém je obvykle konfigurován přes menuconfig nebo obdobný nástroj, možnosti konfigurace Android sestavovacího systému jsou o poznání omezenější. Konfigurace je závislá na množině proměnných nastavených buď dynamicky pomocí příkazů popsaných v předchozí kapitole 1.3.2 nebo staticky v souboru *buildspec.mk*.[29]

Android sestavovací systém se taktéž liší v umístění generovaných object souborů a jiných mezivýsledků kompilace. Na rozdíl od rekurzivního systému nejsou žádné z těchto souborů umístěny ve složkách mezi zdrojovými kódy, ale jsou ve výstupní složce, která je implicitně *out*. To znamená, že příkaz `make clean` je v podstatě to samé jako `rm -rf out/`.[29]

Při kompilaci není nutné kompilovat vždy celé AOSP, ale lze vybírat jednotlivé moduly. Každý modul má v souboru ve svém konfiguračním souboru proměnnou `LOCAL_MODULE`, jejíž hodnota je názvem modulu. Modul services, který obsahuje System Server, jenž bylo nutné v této práci měnit, lze samostatně zkompilovat takto:

```
make -j4 services
```

1.4.1 Konfigurovatelné proměnné

Změnou následujících proměnných prostředí lze konfigurovat proces sestavení. Běžně jsou proměnné prostředí nastaveny buď dynamicky příkazem `lunch` nebo staticky v souboru *buildspec.mk*,[29] v případě potřeby je však možné je nastavit ručně v terminálu.

- `TARGET_PRODUCT` – Umožňuje určit pro jaký cíl AOSP kompilujeme – emulátor, konkrétní zařízení, atd. Jednotlivé možnosti jsou popsány v souborech *AndroidProducts.mk* ve složce *build/target/product/*

v případě emulátoru a v podložkách *devices/* v případě konkrétních zařízení. Některé z možností ve verzi `android-4.1.1_r1`: [29]

  - `full` – generická verze
  - `sdk` – Android SDK (Android Software development kit)
  - `full_grouper` – `full` verze upravená pro Asus Nexus 7

- `TARGET_BUILD_VARIANT` – Udává, které moduly budou nainstalovány. Každý modul obsahuje parametr `LOCAL_MODULE_TAGS`, který může nabývat hodnot: *user*, *debug*, *eng*, *tests*, *optional* a *samples*. Podle nastavení této proměnné budou instalovány takovéto moduly:[2][29]
  - `user` – pouze *user* – limitovaná oprávnění
  - `userdebug` – *user* a *debug* – root oprávnění a základní možnosti debugování
  - `eng` – *user*, *debug* a *eng* – vývojová konfigurace s rozšířenými možnostmi debugování
- `OUT_DIR` – Možnost vybrat složku, do které bude uloženo zkompilované AOSP. Implicitně se jedná o složku *out*.

Mimo tyto základní konfigurační proměnné, které bylo nutné pro potřeby této práce měnit pouze výjimečně, bylo nutné použít některé specifické. Při standardní kompilaci jsou všechny Java aplikace uloženy v tzv. „odex“ formě pro urychlení prvního startu OS. Ke každé apk a jar aplikaci náleží ještě příslušný odex soubor. Mechanismus je podrobněji popsán v kapitole 2.2.4. Zabránit této vlastnosti lze nastavením proměnných:[23]

```
export DISABLE_DEXPREOPT=true
export WITH_DEXPREOPT=false
```

Další proměnou, která se hodí především pro testování ovladačů a funkčnosti nativní části AOSP, je `BUILD_TINY_ANDROID=true`. Vynutí sestavení pouze nativní části AOSP bez Android frameworku.[29] Výhodou je velice rychlá kompilace a malá výsledná velikost sestavení, které je vhodné pro debugování problémů na nejnižší úrovni OS pomocí terminálového připojení.

Kapitola 2

Možnosti úprav OS Android

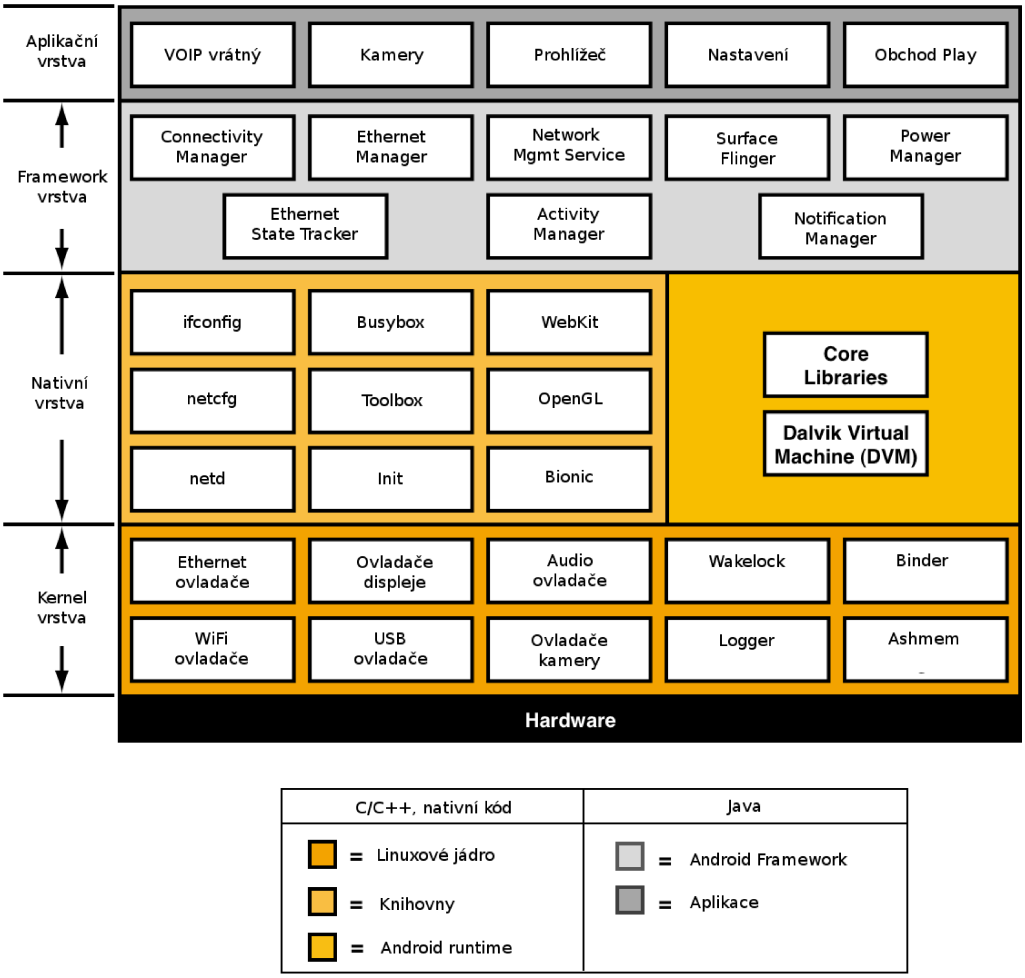
2.1 Architektura OS Android

Pro popis možností úprav OS Android je nedříve potřeba znát jeho architekturu, která se liší od architektury jiných UNIXových systémů. Na diagramu 2.1 je zobrazena architektura OS Android se zaměřením na součásti, které jsou důležité pro tuto práci.[31]

Zjednodušeně lze operační systém Android rozdělit na čtyři vrstvy. Nejnižší vrstvou je linuxové jádro, které bylo pro potřeby mobilního operačního systému výrazně upraveno. Nativní vrstvu tvoří knihovny, aplikace a daemoni, z nichž některé jsou vyvíjeny jako open source projekty známé například z linuxových distribucí a některé jsou specifické pro Android. Součástí nativní vrstvy je i Dalvik Virtual Machine (virtuální stroj určený k interpretaci Java aplikací) a Java knihovny (projekt Apache Harmony). Framework vrstva se skládá především ze systémových služeb, které nabízí aplikační vrstvě API (varianty API zavisí na verzi Androidu, jejichž popis je v tabulce 1.1). Framework vrstva komunikuje s nativní vrstvou pomocí JNI (Java Native Interface).[9][25][29] Samotná aplikační vrstva je důvěrně známá všem, kteří někdy přišli do styku s Android zařízením, jedná se o aplikace, které jsou distribuované například pomocí Google Play.

2.1.1 Kernel vrstva

Jak již bylo zmíněno dříve, Android běží na upraveném linuxovém jádru, které je z důvodu rozsahu úprav nekompatibilní s jádrem, jehož zdrojové kódy jsou zveřejněny na stránce `kernel.org`. Mezi provedené změny patří mimo jiné:[29]



Obrázek 2.1: Architektura OS Android[9][25][29]

- wakelock – detailně popsáno v sekci 2.1.1.1
- binder – RPC/IPC mechanismus – umožňuje komunikaci mezi aplikacemi a framework vrstvou (skryté za volání API) a aplikacemi navzájem
- ashmem – anonymous shared memory – IPC mechanismus, který umožňuje sdílení paměti mezi procesy
- logger – detailně popsáno v sekci 2.1.1.2

Tablet Ainol Novo 7 Crystal s OS od výrobce obsahuje jádro ve verzi 3.0.8.

2.1.1.1 Wakelock

V případě běžného notebooku s linuxem je možné jej na žádost uživatele uspat nebo vypnout z důvodu úspory energie. V případě mobilních zařízení jsou nároky na úsporu energie ještě vyšší a je od nich zpravidla vyžadován provoz 24 hodin denně. Na rozdíl od „uspání“ na požádání uživatele jádro Androidu uspí zařízení pokaždé kdykoliv je to možné.[29]

Při použití tohoto mechanismu je však nutné předejít uspání systému v nevhodný okamžik, jako může být stahování souboru nebo provádění aktualizace. Aplikace si v kritické sekci, kdy nechce být přerušena uspáním systému, vyžádá wakelock a v okamžiku, kdy operaci dokončí, wakelock uvolní. V tabulce 2.1 jsou existující druhy wakelocků a jaký dopad mají na stav CPU a displeje.

Tabulka 2.1: Typy wakelocků[13]

Název	**CPU**	**Displej (podsvícení)**
PARTIAL_WAKE_LOCK	On	Off
SCREEN_DIM_WAKE_LOCK	On	On (ztlumené)
SCREEN_BRIGHT_WAKE_LOCK	On	On (plné)
FULL_WAKE_LOCK	On	On (plné)

Zámky, které udržují zapnutý displej budou zrušeny v případě, že uživatel stiskne tlačítko Power a manuálně tím vypne displej. Naproti tomu zámek PARTIAL_WAKE_LOCK by měl udržovat CPU zapnuté bez ohledu na stav displeje a uživatelské vstupy.[13]

Tablet fungující jako domovní vrátný nesmí nikdy „usnout“, protože je nezbytné, aby byl schopný nepřetržitě komunikovat po síti. Jako řešení se nabízí zámek PARTIAL_WAKE_LOCK, který by neustále držela některá z aplikací, nejlépe VOIP vrátný. Tento postup se ovšem ukázal jako nefunkční a pokaždé když se displej tabletu vypnul, ať už na základě uživatelského vstupu nebo automaticky, přestal tablet komunikovat po síti. Z tohoto důvodu byl namísto toho použit zámek SCREEN_DIM_WAKE_LOCK, který udrží síť neustále spuštěnou, ale za cenu zbytečně spuštěného displeje.

Předpokládá se, že na vině je špatná implementace zámku PARTIAL_WAKE_LOCK v jádře, které je součástí oficiálního sestavení OS Android od Ainolu.

2.1.1.2 Logger

Logování je základní funkcí linuxového systému ve kterém je realizováno dvěma logovacími systémy. Logy jádra jsou přístupné pomocí příkazu `dmesg` a systémové logy jsou běžně ukládány ve složce */var/log/*. Zatímco logy jádra byly ponechány původní, Android přidal vlastní implementaci systémových logů, které více vyhovují specifickým požadavkům mobilních zařízení.[29]

Žádné z logů nejsou ukládány do trvalé paměti, namísto toho jádro udržuje v paměti RAM čtyři kruhové buffery, kde každý z bufferů má svůj záznam ve složce */dev/log/* a pomocí knihovny *liblog* je umožňeno user-space komponentům do logů zapisovat. Jednotlivé buffery jsou určeny k logování ruzných událostí a jsou následující:[29]

- events – systémové události
- main – hlavní log používaný aplikacemi
- radio – logy obsahující informace o síti (GSM, 3G, LTE)
- system – systémový log

Logy lze vypsat příkazem `logcat -b events`, kde přepínač `-b` udává název bufferu, který má být vypsán (implicitní hodnota je main). Při práci na systémové úrovni mohu z vlastní zkušenosti doporučit sledovat všechny logy, protože různé součásti OS logují různé události různě. V případě Ainol tabletu není třeba sledovat buffer radio, neboť tablet nemá hardware pro použití mobilní sítě.

2.1.2 Nativní vrstva

2.1.2.1 Init

Stejně jako v linuxovém systému je `init` prvním procesem, který je spuštěn jádrem. Ten následně inicializuje zbytek systému. Obvykle distribuce linuxu používají SystemV init. Proces `init` využívaný v OS Android byl vytvořen speciálně pro Android, využívá svojí vlastní syntaxi konfiguračních souborů a je schopný reagovat na změny *global properties*.[29]

Global properties jsou dvojice typu klíč: hodnota, které jsou dostupné napříč OS. Jejich seznam je možné vypsat příkazem `getprop` a nastavit je lze pomocí `setprop <key> <value>`, v případě, že daný klíč neexistuje, bude vytvořen. *Global properties* jsou nastavené při startu systému ze souborů */default.prop* a */system/build.prop* a následně jsou měněny dynamicky OS nebo staticky uživatelem.

Hlavním, a v případě Ainol Novo 7 Crystal jediným, konfiguračním souborem procesu `init` je soubor */init.rc*. Detailní popis jeho konfiguračních možností není cílem této práce, ale dovolím si uvést alespoň příkazy, které jsem prakticky využil.

```
service netup /system/bin/netup
  class main
  disabled
  oneshot
```

Definuje novou službu `netup`, která je součástí třídy `main`, implicitně je vypnutá (`disabled`) a při skončení se jí `init` nepokouší znovu spustit (`oneshot`).[29] Při zavolání příkazu `start netup` bude `/system/bin/netup` spušťeno procesem `init` na pozadí.

```
on property:start.netup=1
  start netup
```

Zajistí, že v případě změny global property start.netup na 1 proces `init` spustí službu `netup`. V tomto případě tedy příkazy `start netup` a `setprop start.netup 1` budou mít stejný výsledek.

2.1.2.2 Filesystém

Android filesystém funguje obdobně jako v případě linuxu na principu root filesystému, do kterého jsou ostatní součásti mountovány. Zásadně se ovšem liší v adresářové struktuře. Některé z důležitých součástí Android filesystému jsou popsány níže.

Samotná neměnná data OS, která vznikla sestavením ze zdrojového kódu se nachází ve složce */system*, která je mountovaná jako samostatný diskový oddíl. Zde se nachází nativní daemoni a aplikace (kromně `/init` , `/sbin/adbd` a `/sbin/ueventd`), knihovny, Android framework a systémové aplikace (ty, které nejdou standardně odinstalovat).

Další důležitou součástí je složka */data*, která obsahuje uživatelem doinstalované aplikace a nastavení. Složka se rovněž nachází na samostatném diskovém oddílu a jeho naformátování de facto znamená uvedení zařízení do továrního nastavení.

Složka */cache* je rovněž mountována ze samostatného diskového oddílu a její význam je popsán v sekci 4.4

Složky */dev*, */proc* a */sys* mají obdobné funkce jako stejné složky v linuxovém systému.

2.1.2.3 Nativní daemoni

Součástí OS Android je i několik nativních daemonů, které jsou spouštěny přes `init`. Z hlediska této práce je důležitý pouze `/system/bin/netd`, který slouží jako síťový manager[29] a `/system/bin/dhcpcd`, který zajišťuje získávání konfigurace sítě přes protokol DHCP.

2.1.2.4 Nativní aplikace

Android ve složce */system/bin* a */system/xbin* mimo daemonů obsahuje i standardní aplikace známé z linuxového prostředí nebo alespoň aplikace jim podobné. Pro konfiguraci sítě je možné využít příkazy `netcfg`, `ifconfig` a `route`. Poslední dva jsou pouze symbolický link do binárky `toolbox`, která funguje obdobně jako dobře známý BusyBox, má pouze omezenější možnosti. BusyBox standardně není součástí AOSP, ale je možné jej snadno doinstalovat. Ainol dokonce dodává svůj oficiální OS pro tablet s přibaleným BusyBoxem.

2.1.3 Framework vrstva

Zjednodušený, avšak pro potřeby této práce dostačující, pohled na framework vrstvu je jako na soubor služeb OS, kde každá služba má na starost určitou vlastnost OS. Například Notification Manager má na starost zobrazování notifikací uživateli. Pokud chce aplikace zobrazit notifikaci, požadá Notification Managera prostřednictvím API o zobrazení notifikace a ten se o to postará.[12] K docílení požadovaného efektu systémové služby komunikují spolu navzájem a také s nižšími vrstvami OS. Služby OS nereagují pouze na požadavky, které jim aplikace předá pomocí API, ale i na události jako vypnutí/zapnutí displeje nebo připojení USB zařízení.

Vypsat všechny služby OS Android lze příkazem `service list`, v případě Androidu verze 4.1.1 se jedná o 67 služeb. Na každé z těchto 67 služeb mohou záviset ostatní služby nebo ostatní součásti OS, není tedy snadné je příliš upravovat nebo dokonce některou z nich odstranit. Situaci zhoršuje fakt, že zdrojové kódy jsou pouze řídce komentované a neexistuje k nim žádná dokumentace.[29]

Zdrojové kódy systémových služeb se v rámci AOSP nachází ve složce *frameworks/base/services/*.

2.1.4 Aplikační vrstva

Aplikační vrstva obsahuje dva druhy aplikací, systémové a uživatelské. Systémové aplikace jsou součástí OS a nelze je standardně odinstalovat. Jedná

se například o základní Prohlížeč, Nastavení a Google Play aplikaci. Jejich APK soubory jsou uloženy ve složce */system/app*. APK nebo-li Application package file je balíček (archiv), pomocí kterého jsou distribuovány aplikace na platformě Android.

Uživatelské aplikace jsou ty, které si uživatel doinstaluje sám například přes Google Play nebo příkazem `install`. APK soubory těchto aplikací jsou uloženy ve složce */data/app*. Pro zajištění funkce tabletu jako domovního vrátného obsahuje následující předinstalované uživatelské aplikace:

- VOIP Vrátný – umožňuje videohovor s osobou u vchodu a vzdálené otevření vchodových dveří
- Kamery – umožňuje sledovat kamery instalované v domě
- DroidSSHd – SSH server
- Wake Lock – drží SCREEN_DIM_WAKE_LOCK, při vyřešení problému PARTIAL_WAKE_LOCK je možné ji odinstalovat, protože PARTIAL_WAKE_LOCK drží i aplikace VOIP Vrátný

2.2 Ainol Novo 7 Crystal — možnosti úprav

Hlavní parametrem určujícím možnosti úprav OS Android je dostupnost zdrojových kódů pro dané zařízení a možnost zápisu provedených změn do zařízení. Přestože AOSP je open source, není na většině zařízení možné jej použít tak jak je a jsou nutné rozsáhlejší úpravy. Ty obvykle udělá výrobce hardwaru, v tomto případě Amlogic (SoC), který se v daných zařízeních nachází. Výrobce tabletů (Ainol) poté zdrojové kódy může dále upravovat tak, aby zajistil podporu veškerému hardwaru, který se v tabletu mimo SoC nachází. Možnosti zápisu provedených změn do zařízení jsou v našem případě neomezené, detailní popis této vlastnosti se nachází v sekci 4.3.

Nic nenutí výrobce tabletů zveřejňovat zdrojové kódy svých úprav. Pokud je neuvolní, což je ve světě Androidu běžná praxe, jsou možnosti úpravy OS značně omezené. Jak je již popsáno v sekci 1.2.1, pokusili jsme se společně s vedoucím práce získat zdrojové kódy přímo od výrobce SoC, ovšem bezvýsledně.

Ainol zdrojové kódy pro zařízení Novo 7 Crystal neuvolnil a bude tedy nutné najít způsob, jak se s tímto omezením vypořádat. Během práce bylo zjištěno, že závada se nachází ve framework vrstvě, následující popis možností úprav se tedy bude zaměřovat primárně na tuto část OS.

2.2.1 Sestavení ze zdrojových kódů Ainol

Ainol uvolnil zdrojové kódy včetně jádra pro zařízení Elf II, které má totožný SoC jako Crystal. Jedná se o zdrojové kódy Androidu verze 4.0.3, zatímco v Crystalu je Android ve verzi 4.1.1. Na framework vrstvě je zajištěna zpětná kompatibilita pouze na API, které je dostupné aplikační vrstvě, ale ne ve způsobu, kterým systémové služby komunikují se zbytkem OS a spolu navzájem. Výměna systémových služeb v Androidu 4.1.1 za služby z Androidu 4.0.3, tedy může být problematická. Zdrojové kódy jsou ke stažení na adrese `http://bbs.ainol.com/forum.php?mod=viewthread&tid=16354&extra=page%3D1`.

2.2.2 Sestavení ze zdrojových kódů Google

K upravení OS je možné použít originální zdrojové kódy AOSP, které lze zíkat podle popisu v sekci 1.2.2. Podle popisu v sekci 1.4 lze sestavit pouze modul, který jsme potřebovali upravit a tím následně nahradit neupravenou součást OS v tabletu. Tento postup předpokládá, že způsob, kterým upravovaná komponenta komunikovala se zbytkem systému, nebyl úpravami výrobce tabletu změněn.

2.2.3 Úpravy bez použití zdrojových kódů

Možnosti úprav OS Android jsou v tomto případě velmi omezené a vztahují se především k nativní vrstvě OS. Jednou z možností je upravit chování procesu `init` prostřednictvím změny konfiguračního souboru */init.rc*. Je možné zde definovat vlastní služby a kdy budou spouštěny nebo změnit vlastnosti aktuálních služeb (viz 2.1.2.1). Služby definované v */init.rc* nemají nic společného se systémovými službami, které běží na framework vrstvě OS. Pomocí procesu `init` je možné spouštět pouze nativní aplikace a daemony.

Binárku `/init` a její konfigurační soubor */init.rc* stejně jako jádro systému nelze měnit přímo na spuštěném tabletu. Jsou součástí bootovatelného obrazu */boot.img*, který je součástí originálního aktualizačního balíčku od Ainolu. Detailní popis aktualizačních balíčku je v sekci 4.2. Soubor je možné rozbalit a opět zabalit pomocí nástrojů, které jsou ke stažení na `https://disk.yandex.com/public/?hash=oc1f9iWZwK8Bbcs6Fzuq72yZ9EpoZFWkk4hzsBWqFMk%3D`. Tato sada skriptů je určema pro použití na OS Windows, při použití na Linuxu jsou relevantní pouze skripty *bin/uImage_recovery_repack.sh* a *bin/uImage_recovery_unpack.sh*. Pro rozbalení slouží příkaz

`uImage_recovery_unpack.sh boot.img`. Jeho provedením vznikne složka *initramfs*, která obsahuje jak binárku `init` tak i konfigurační soubor *init.rc*. Po jeho upravení lze soubory zabalit zpět pomocí příkazu `uImage_recovery_repack.sh boot.img`. Soubor *uImage-tmp/boot.img.new* je nový bootovatelný obraz obsahující upravenou verzi */init.rc*.

Ostatní konfigurační soubory se nachází ve složce */system/etc*, ale jejich možnosti jsou značně omezené a v průběhu práce jsem je měnit nepotřeboval.

Další možností je vytvořit vlastní skript nebo program, který bude ovlivňovat chování OS. Skripty je možné psát standardně jako na linuxu s výjimkou řádky určující cestu k interpreteru, ta musí být na androidu: `#! /system/bin/sh`. Vytvořit vlastní program v C je možné díky Android NDK (Native Development Kit) následovně:[24]

1. Stáhnout NDK na `http://developer.android.com/tools/sdk/ndk/index.html`
2. Vytvořit požadovaný toolchain: `android-ndk-r9b/build/tools/make-standalone-toolchain.sh --arch=arm --ndk-dir=android-ndk-r9b --install-dir=android-toolchain --platform=android-16 --system=linux-x86_64`
3. Příkazem `android-toolchain/bin/arm-linux-androideabi-gcc` je nyní možné kompilovat programy napsané v C určené pro nativní běh na tabletu

2.2.4 Reverzní inženýrství

Reverzní inženýrství Java aplikací, které jsou součástí frameworku hrálo při řešení této práce klíčovou roli. Je díky němu možné, leč omezeně, ovlivnit fungování systémových služeb.

2.2.4.1 Android Java

Standardní Java se skládá ze tří základních součástí:[29]

- Java kompilátor – překládá zdrojové kódy do Java bytekódu (*class* soubory)
- Java virtuální stroj – interpreter Java bytekódu
- Knihovny

OS Android sám o sobě neobsahuje ani jednu z těchto součástí. Pro interpretaci Java aplikací používá Dalvik VM, vlastní implementaci virtuálního stroje, a namísto standardních Java knihoven obsahuje Apache Harmony, open source reimplementaci těchto knihoven. Java kompilátor je využíván při kompilaci zdrojových kódů Javy, ale jako takový není součástí OS. Dalvik VM neinterpretuje přímo *class* výstupní soubory kompilátoru, ale soubory *dex*, které vzniknou dalším zpracováním *class* souborů utilitou `dx`.[29] Pro urychlení prvního startu OS jsou aplikace sestavovacím systémem generovány v takzvaném „odex" formátu, kdy ke každé aplikaci ve formě jar nebo apk balíčku náleží jeden *odex* soubor.[29]

2.2.4.2 Rozbalení aplikace na úroveň Java bytekódu

Postup pro rozbalení aplikace do Java bytekódu bude popsán na příkladu rozbalení framework aplikace *services.jar*, která se v originálním OS od Ainolu nachází ve složce */system/framework* v „odex" formátu, tzn. skládá se ze dvou souborů *services.jar* a *services.odex*.

Pro „deodexování" aplikace jsou potřeba nástroje `baksmali.jar` a `smali.jar`, které jsou ke stažení na stránce `https://bitbucket.org/JesusFreke/smali/downloads`. Převedení souboru *services.odex* na *classes.dex* se provede následovně:

```
$ java -jar baksmali.jar -a 16 -d framework \
> -x framework/services.odex
$ java -jar smali.jar -a 16 -o classes.dex out
```

Přepínač `-a` udává úroveň API (viz tabulka 1.1), přepínač `-d` udává cestu ke složce ve které se nachází všechny *odex* soubory frameworku a přepínač `-x` udává který soubor má být „deodexován". Program musí mít přístup k ostatním *odex* souborům frameworku, protože jednotlivé *odex* soubory nenesou informaci celé aplikace, ale odvolávají se na sebe navzájem. Proto je nezbytné, i při potřebě „deodexovat" pouze jediný soubor, „deodexovat" všechny, které se v OS nachází. Konkrétně se jedná o soubory ve složkách */system/framework* a */system/app*. Výstupem prvního příkazu je složka out ve které je rozbalený soubor *services.odex*. Druhý příkaz z této složky vytvoří soubor *classes.dex*, který je spustitelný na Dalvik VM.

Nyní stačí přidat soubor *classes.dex* do jar archivu například příkazem `zip -r framework/services.jar classes.dex` a proces „deodexace" aplikace je tím dokončen.

K rozbalení archivu *services.jar* na jednotlivé *class* soubory slouží sada nástrojů `dex2jar`, která je ke stažení na adrese `http://code.google.com/p/dex2jar/downloads/list`. Postup je následující:[8]

```
$ d2j-dex2jar.sh -f -o unpacked.jar services.jar
$ d2j-asm-verify.sh unpacked.jar
$ d2j-jar2jasmin.sh -f -o bytecode unpacked.jar
```

První příkaz zajistí převedení archivu *services.jar*, který obsahuje pouze jeden soubor *classes.dex* na archiv *unpacked.jar*, který obsahuje jednotlivé *class* souboy. Druhý příkaz provede kontrolu správnosti archivu *unpacked.jar*. Poslední příkaz rozbalí archiv *unpacked.jar* do složky *bytecode* a jednotlivé *class* soubory převede na textové soubory obsahující Java bytekód.

K prohlížení zdrojových kódu aplikace se mi osvědčil program `Java Decompiler`, který je ke stažení na stránce `http://jd.benow.ca/`. Samotné změny je třeba provádět přímo na úrovni Java bytekódu v souborech ve složce *bytecode* například editorem `Geany`.

2.2.4.3 Opětovné zabalení aplikace

Po provedení požadovaných změn je třeba aplikaci opět zabalit do archivu *services.jar*, která bude obsahovat pouze soubor *classes.dex*. Postup je následující:[8]

```
$ d2j-jasmin2jar.sh -f -o new_unpacked.jar  bytecode/
$ d2j-asm-verify.sh new_unpacked.jar
$ d2j-jar2dex.sh  -f -o classes.dex new_unpacked.jar
$ zip -r services.jar classes.dex
```

První příkaz převede soubory s bytekódem ze složky *bytecode* zpět na jar archiv *new_unpacked.jar* obsahující *class* soubory. Druhý příkaz provede kontrolu správnosti tohoto archivu. Třetí příkaz převede tento archiv na soubor *classes.dex*, který je interpretovatelný pomocí Dalvik VM. Poslední příkaz jej přidá do archivu *services.jar*.

Při následné instalaci této aplikace do tabletu se změny provedené na úrovni Java bytekódu projeví v jejím chování.

Kapitola 3

Chybná implementace zámku PARTIAL_WAKE_LOCK

3.1 Detailní popis závady

Tablet Ainol Novo 7 Crystal je instalován v bytových jednotkách, kde slouží jako domovní vrátný. Především musí zajistit možnost videohovoru s osobou stojící u zvonků u vchodových dveří domu. Komunikace tabletu se zvonky je realizována pomocí sítě ethernet, ke které je tablet připojen přes USB adaptér Edimax EU-4208.

Aplikace `VOIP Vrátný`, která zajišťuje komunikaci se zvonky u vchodových dveří, drží trvale zámek PARTIAL_WAKE_LOCK, což by mělo zabránit jádru OS tablet uspat. Mechanismus je detailně popsán v sekci 2.1.1.1.

Tablet se sestavením OS Android od výrobce přestane při vypnutí displeje komunikovat po ethernet síti a tím pádem plnit svou funkci. Při následném zapnutí displeje tablet začne opět komunikovat přes toto rozhraní. Jedním z cílů této práce byla úprava OS Android, která umožní nepřetržitou komunikaci tabletu po ethernet rozhraní nezávislou na stavu displeje. Zadání vycházelo z předpokladu, že v tomto sestavení nefunguje korektně zámek PARTIAL_WAKE_LOCK a tablet i přes jeho držení aplikací `VOIP Vrátný` při vypnutí displeje usne.

Do té doby než byla závada odstraněna byla v tabletu nainstalována aplikace `Wake Lock - PowerManager` (`https://play.google.com/store/apps/details?id=eu.thedarken.wl&hl=en`). Aplikace byla nakonfigurována k držení zámku SCREEN_DIM_WAKE_LOCK, který udržoval neustále zapnutý displej a tím pádem jistotu, že tablet nemůže usnout. Podstatnou nevýhodou tohoto řešení byl právě neustále zapnutý displej, který

nejenže v bytě působil rušivě, ale zbytečně se opotřebovával více než bylo nutné. Po odstranění závady je možné tuto aplikaci odinstalovat a umožnit tak vypínání displeje během provozu tabletu.

3.2 Testování funkčnosti PARTIAL_WAKE_LOCK

Na počátku práce se předpokládalo, že tablet nekomunikuje s okolím při vypnutém displeji z důvodu špatné implementace zámku PARTIAL_WAKE_LOCK. Tablet by v takovém případě přešel při zhasnutí displeje do režimu spánku (tzv. „deep sleep“) a jako důsledek by přestal komunikovat po síti.

Tento předpoklad byl ověřen pomocí aplikace `Wakelock Detector` (`https://play.google.com/store/apps/details?id=com.uzumapps.wakelockdetector&hl=en`). Aplikace poskytuje informace o ostatních aplikacích držících nějaký wakelock a souhrnnou informaci o tom, jak dlouho se tablet nacházel v režimu spánku (tzv. „deep sleep“). Bylo provedeno několik měření, při kterých byl tablet vystaven různým situacím, které mohou nastat. Jednalo se o pravidelné manuální i automatické zapínaní a vypínání displeje, nabíjení tabletu pomocí externí nabíječky i pomocí USB a připojování a odpojování ethernet adaptéru. Tato měření ukázala, že tablet se při držení zámku PARTIAL_WAKE_LOCK nikdy nedostane do režimu spánku a naopak při jeho uvolnění usne podle předpokladů.

Tento fakt je možné pozorovat i ve výpisu logu jádra `dmesg`. Při přechodu OS do režimu spánku je zde možné pozorovat jednotlivé kroky, které OS vykonává při přechodu do režimu spánku a následnou zprávu o spánku zařízení. Některé z nich jsou uvedeny v ukázce níže. Jednotlivé logované zprávy jsou uváděny chronologicky, ale mezi nimi se nachází i další zprávy, které jsou v ukázce vynechány.

```
<4>[ 2011.630023@1] Freezing user space processes ...
<4>[ 2011.670258@1] Freezing remaining freezable tasks ...
<6>[ 2012.887690@1] PM: suspend of devices complete ...
<4>[ 2012.903716@1] Disabling non-boot CPUs ...
<6>[ 2012.986058@0] sleep ...
```

Pokud některá z aplikací drží zámek PARTIAL_WAKE_LOCK, žádné z těchto událostí nikdy nenastanou a OS při zhasnutém displeji nadále

běží. Implementace zámku PARTIAL_WAKE_LOCK je tedy v případě OS běžícího v tabletu správná.

Tablet přesto při vypnutí displeje přestane komunikovat po ethernet síti a plnit funkci domovního vrátného. Důvody tohoto chování je třeba hledat v jiné části OS.

3.3 Problém EthernetManagera

Oficiální Android AOSP neobsahuje podporu připojení pomocí USB ethernet adaptéru. Tato funkčnost musí být do zdrojových kódů, které Google poskytuje, přidána výrobcem zařízení. Jednou ze součástí, kterou musí vývojář do oficiálních zdrojových kódů přidat, je systémová služba Ethernet Manager.[31]

Ethernet Manager má na starosti správu ethernet připojení, které má zařízení k dispozici. Aplikace s ním mohou komunikovat na základě API a Ethernet Manager následně komunikuje se zbytkem OS k zajištění správného fungování ethernet spojení.

Ainol v případě tabletu Elf II, ke kterému uvolnil zdrojové kódy, použil implementaci Ethernet Managera z projektu „The Android-x86 Open Source Project“. Tato implementace obsahuje vlastnost, kdy při zhasnutí displeje Ethernet Manager vypne ethernetová rozhraní. Část implementace Ethernet Managera obsahující tuto vlastnost je následující:

```
private static final String TAG = "EthernetService";

private final BroadcastReceiver mReceiver =
                new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
       String action = intent.getAction();
       if (action.equals(Intent.ACTION_SCREEN_ON)) {
         Slog.d(TAG, "ACTION_SCREEN_ON");
         mDelayedHandler.postDelayed(mResetInterface, 5*1000);
       } else if (action.equals(Intent.ACTION_SCREEN_OFF)) {
         Slog.d(TAG, "ACTION_SCREEN_OFF");
         mDelayedHandler.removeCallbacks(mResetInterface);
         mTracker.stopInterface(false);
       }
       }
    };
```

Jak je vidět Ethernet Manager čeká na intent udávající změnu stavu displeje a v závislosti na stavu displeje vypíná a zapíná v tabletu ethernet rozhraní. Jedná se o implementaci pro Elf II, která je však v případě tabletu Crystal obdobná. Pro ověření lze prohlédnout události v systémovém logu tabletu Crystal při změnách stavu displeje příkazem `logcat -b system`. Události v systémovém logu při vypnutí displeje:

```
D/EthernetService( 3366): ACTION_SCREEN_OFF
I/EthernetStateTracker( 3366): stop dhcp and interface
V/EthernetStateTracker( 3366): report new state DISCONNECTED
                               on dev usbnet0 current=usbnet0
V/EthernetStateTracker( 3366): update network state tracker
I/EthernetStateTracker( 3366): Old status stackConnected=true
                               HWConnected=true
I/EthernetStateTracker( 3366): [EVENT: ether is removed]
D/EthernetStateTracker( 3366): setEthState state=true->false
                               event=4
D/EthernetStateTracker( 3366): ***isConnected: true
D/EthernetStateTracker( 3366): ***isConnected: false
D/EthernetStateTracker( 3366): EVENT_HW_DISCONNECTED:
                               StopInterface
I/EthernetStateTracker( 3366): stop dhcp and interface
I/EthernetStateTracker( 3366): New status, stackConnected=true
                               HWConnected=false
D/ConnectivityService( 3366): ConnectivityChange for ethernet:
                              DISCONNECTED/DISCONNECTED
D/ConnectivityService( 3366): Attempting to switch to mobile
D/ConnectivityService( 3366): Attempting to switch to wifi
D/ConnectivityService( 3366): resetConnections(usbnet0, 3)
```

Tablet tedy nemůže komunikovat při zhasnutém displeji pomocí ethernet rozhraní kvůli nevhodné implementaci systémové služby Ethernet Manager, zatímco implementace zámku PARTIAL_WAKE_LOCK je v pořádku.

3.4 Možnosti odstranění

Z důvodu nedostupnosti zdrojových kódu k sestavení Androidu, který běží v tabletu Ainol Novo 7 Crystal, bylo potřeba najít jinou cestu, jak změnit fungování systémové služby Ethernet Manager. Celkem byly rozpracovány čtyři varianty možného řešení problému, kde pouze reverzní inženýrství nakonec vedlo k požadovaným výsledkům.

3.4.1 Daemon netup

Vůbec první možností jak problém vyřešit bylo napsat deamona, který obejde Ethernet Managera a bude na nativní úrovni OS ethernet rozhraní spravovat sám. Daemon zajišťuje spuštění rozhraní a jeho konfiguraci přes DHCP. Nevýhodou tohoto řešení je, že původní Ethernet Manager je neustále aktivní. Při každém vypnutí displeje vypne Ethernet Manager rozhraní, které následně tento daemon ihned znovu zapne. Z toho důvodu při každém vypnutí displeje nastane na rozhraní rychlý přechod up/down.

Součástí řešení je nativní program `watchdog`, který pasivně čeká na změnu v globální proměnné „init.svc.dhcpcd_usbnet0“ a v případě její změny se ukončí. Tato proměnná se změní pokaždé, když se Ethernet Manager pokusí vypnout rozhraní usbnet0 (USB ethernet adaptér). Skript `netup` v nekonečné smyčce provádí následující úkony:

- spustí program `watchdog`
- spustí rozhraní a nastaví mu správné adresy

Samotné nastavení rozhraní lze provést těmito příkazy:

```
# netcfg usbnet0 up
# ifconfig usbnet0 192.168.0.3 netmask 255.255.255.0
# route add default gw 192.168.0.1 dev usbnet0
```

Ethernet Manageru je třeba také zabránit ve spouštění DHCP klienta na tomto rozhraní, neboť to bude mít na starost daemon `netup`. Dosáhnout toho lze pomocí skriptu `infinite_loop`, který pouze v nekonečné smyčce volá příkaz `sleep 3600`. Ethernet Manager spouští DHCP klienta pomocí procesu `init`. DHCP klient je v jeho konfiguračním skriptu */init.rc* definován jako služba `dhcpcd_usbnet0` takto:

```
service dhcpcd_usbnet0 /system/bin/dhcpcd -ABKL
class main
group dhcp system
disabled
oneshot
```

Při změně první řádky této deklarace na `service dhcpcd_usbnet0 /system/bin/infinite_loop -ABKL`, se při každém pokusu o spuštění nebo ukončení DHCP klienta Ethernet Managerem pouze spustí nebo ukončí skript `infinite_loop`.

Vzhledem k tomu, že skript `netup` spouští DHCP klienta přes proces `init`, je třeba definovat novou službu v */init.rc* pro jeho spouštění a také vypnutí při vypnutí skriptu `netup` následovně:

```
service dhcpcd_new /system/bin/dhcpcd -ABKL
class main
disabled
oneshot

on property:init.svc.netup=stopped
stop dhcpcd_new
```

Dále již stačí pouze definovat skript `netup` jako službu, která bude spouštěna vždy při startu OS takto:

```
service netup /system/bin/netup
class main
oneshot
```

Toto řešení funguje, ovšem pouze pro součásti OS, které běží v nativní vrstvě. Tablet odpovídá na `ping` a je možné se na něj připojit pomocí SSH. Ethernet Manager však nemá informaci o tom, že je tablet připojen k síti, neboť síťové připojení nenastavil on, ale skript `netup`. Všechny součástí OS, které jsou ve framework a aplikační vrstvě OS tudíž informuje o tom, že tablet je odpojen od sítě, čímž jim znemožňuje komunikaci.[31] Aplikace VOIP vrátný běží v aplikační vrstvě OS a toto řešení se pro ni tedy ukázalo jako nepoužitelné.

3.4.2 Zkompilování zdrojových kódů Ainol

Druhou možností řešení, která byla v průběhu práce vyzkoušena bylo použití zdrojových kódů k tabletu Elf II, které Ainol uvolnil. Tento tablet má oproti tabletu Crystal některý hardware odlišný. Z tohoto důvodu se nepodařilo použít tyto zdrojové kódy jako celek na model Crystal.

Problém by bylo možné vyřešit i v případě, že se podaří zkompilovat pouze modul AOSP, který obsahuje Ethernet Managera, a následně jím nahradit daný modul v tabletu. Modul, který obsahuje většinu systémových služeb včetně Ethernet Managera se nazývá *services*.

Zásadní problém při použití tohoto postupu je nezaručená zpětná kompatibilita mezi systémovými službami.[29] Zdrojové kódy, které Ainol uvolnil jsou ve verzi 4.0.3, zatímco v tabletu běží Android ve verzi 4.1.1. Ve verzi 4.0.3 modul *services* obsahuje 53 systémových služeb, zatímco ve verzi 4.1.1 jejich počet narostl na 58. Některé při nahrazení modulu *services* v tabletu modulem ze starší verze chybí a některé jsou nekompatibilní se zbytkem OS, což způsobuje neschopnost OS nabootovat. Samotný modul *services* obsahuje ve verzi 4.0.3 4.8 MB nedokumentovaných zdrojových kódů, ve

verzi 4.1.1 je to již 5,4 MB. Proto jsou jakékoliv rozsáhlejší úpravy těchto zdrových kódu velmi složité.

Z těchto důvodů se využití tohoto postupu v průběhu práce ukázalo jako nereálné.

3.4.3 Zkompilování zdrojových kódů Google

Třetí možností je využití originálních zdrojových kódů nabízených Googlem. Výhodou tohoto postupu oproti využití zdrojových kódů zveřejněných Ainolem je možnost vybrat stejnou verzi AOSP jako verze běžící v tabletu. Pro potřeby této práce jsem využil AOSP ve verzi android-4.1.1_r1.

Nevýhodou je to, že součástí oficiálních zdrojových kódů od Googlu není Ethernet Manager, je tedy třeba provést patch zdrojových kódů. Další problém teoreticky nastane v případě, pokud Ainol změní způsob jakým jednotlivé součásti OS komunikují s modulem *services*. Takto vzniklé chyby jsou těžko odstranitelné, právě kvůli nemožnosti zjistit jaké změny Ainol ve zdrojových kódech provedl a absenci dokumentace zdrojových kódů.

Během práce se podařilo provést patch zdrojových kódů do fáze, kdy bylo možné celé AOSP zkompilovat. Výsledný OS však nebylo možné nabootovat právě z důvodu chyb na framework vrstvě, které se sice dařilo postupně odstraňovat, ale problém byl nakonec rychleji vyřešen použitím reverzního inženýrství.

3.4.4 Reverzní inženýrství

Čtvrtou možností řešení, která jako jediná vedla k plně funkčnímu odstranění problému vypínání ethernet sítě při vypnutí displeje je reverzní inženýrství. Tento postup využívá rozbalení modulu *services* až na úroveň Java bytekódu podle popisu v sekci 2.2.4 a následnou úpravu bytekódu s cílem změny fungování tohoto modulu.

Následný postup předpokládá, že je k dispozici soubor *unpacked.jar* a složka *bytecode* vytvořená podle postupu v sekci 2.2.4. Soubor *unpacked.jar* je možné prohlížet přímo na úrovni zdrojových kódů Java pomocí programu Java Decompiler. Při prohlížení třídy EthernetService, která je implementací Ethernet Managera je zřejmé, že implementace části způsobující vypínání rozhraní při vypnutí displeje je stejná jako v případě zdrojových kódů pro Elf II (popsáno v sekci 3.3).

Mezi soubory obsahující Java bytekód je tato funkcionalita implemetována v souboru *bytecode/com/android/server/EthernetService$2.j*. Konkrétně se jedná o následující část:

```
.method public onReceive(Landroid/content/Context;
                         Landroid/content/Intent;)V
aload 2
invokevirtual android/content/Intent/getAction()
                         Ljava/lang/String;
astore 3
aload 3
ldc "android.intent.action.SCREEN_ON"
invokevirtual java/lang/String/equals(Ljava/lang/Object;)Z
ifeq L0
ldc "EthernetService"
ldc "ACTION_SCREEN_ON"
...
...
...
invokevirtual android/net/ethernet/EthernetStateTracker/
                         stopInterface(Z)Z
pop
return
.limit locals 4
.limit stack 4
.end method
```

Jedná se o metodu, která způsobuje změny stavu ethernet rozhraní v závislosti na stavu displeje. Pokud chceme zabránit těmto změnám a udržet ethernet rozhraní neustále zapnuté, stačí jednoduše nahradit obsah metody příkazem `nop`, který v Java bytekódu, stejně jako v assembleru, znamená instrukci, která nevykoná nic. Metoda bude poté vypadat následovně:

```
.method public onReceive(Landroid/content/Context;
                         Landroid/content/Intent;)V
nop
return
.limit locals 4
.limit stack 4
.end method
```

Bytekód je nyní nutné převést zpět na spustitelnou aplikaci pomocí postupu popsaném v sekci 2.2.4. Při zpětném převodu vznikne soubor *new_unpacked.jar*, který lze prohlížet programem Java Decompiler stejně jako soubor *unpacked.jar*. V implementaci třídy EthernetService je možné pozorovat změnu, kdy tělo metody onRecieve je nyní prázdné:

```
private final BroadcastReceiver mReceiver =
                                          new BroadcastReceiver()
{
  public void onReceive(Context paramAnonymousContext,
                                       Intent paramAnonymousIntent)
  {
  }
};
```

Metoda bude sice nadále volána při každé změně stavu displeje, ale vzhledem k tomu, že je prázdná, nezmění stav ethernet rozhraní.

3.5 Dosažené výsledky

Soubor *services.jar*, který vznikl převedením námi upravených souborů s Java bytekódem zpět na spustitelnou aplikaci, nyní obsahuje Ethernet Managera, který funguje podle požadavků.

K otestování funkčnosti je třeba nejdříve aktualizovat OS tabletu podle popisu v kapitole 4. Ethernet Manager nyní funguje korektně. Ethernet rozhraní je neustále spuštěno bez ohledu na změny stavu displeje.

Funkčnost byla testována nejdříve v prostředí lokální sítě, kdy byl tablet po dobu 48 hodin připojen přes switch k počítačí. Po celou dobu byla konektivita tabletu testována příkazem `ping` spuštěným na počítači a trvalým SSH připojením z počítače do tabletu. Tablet byl vystaven událostem typickým pro prostředí ve kterém je provozován. Jedná se o náhodné vypínaní a zapínaní displeje, jak uživatelem, tak automaticky, občasné používání aplikací, které využívají komunikaci po síti a vynucený restart každých 24 hodin. Implementované řešení fungovalo ve všech případech korektně.

Po důkladném otestování v podmínkách lokální sítě byla aktualizace dne 10. 12. 2013 vzdáleně aplikována na jeden z tabletů v domě a do dne, kdy byla dopsána tato práce funguje bezproblémově. V blízké budoucnosti bude aktualizace provedena na zbytku tabletů instalovaných v domě.

Kapitola 4

Vzdálená aktualizace OS

4.1 Lokální aktualizace OS Android

V případě, že je tablet Ainol využíván ke standardním účelům, není obvykle potřeba jej nijak spravovat a pouze výjimečně může být užitečné aktualizovat operační systém. Toto je potřeba obvykle v případě, kdy výrobce uvede aktualizaci odstraňující některé z chyb obsažených v jejich předchozím sestavení OS Android.

Postup aktualizace OS Android na tabletu Ainol Novo 7 Crystal je následující:[19]

1. Nahrejte na MicroSD kartu příslušný firmware (ve formě aktualizačního balíčku)
2. Restartujte tablet do recovery módu (vypnutí a jeho následné zapnutí současným držením tlačítka Power a Vol-)
3. Proveďte `wipe data/factory reset`, `wipe cache partitions` a `wipe media partition`
4. Zvolte `apply update from EXT` a zvolte příslušný soubor s aktualizací (může být vykonáno vícekrát pokud je takových souborů více)
5. Zvolte `reboot system now`

Výše popsaný postup aktualizace je vyhovující a plně funkční při běžném použití. Pro použití k aktualizaci tabletu, který je využíván v bytové jednotce klienta jako domovní vrátný, je ale ze dvou důvodů naprosto nevyhovující. Prvním důvodem, který zabraňuje použití tohoto postupu je to, že při něm lze použít pouze aktualizační balíčky, které jsou správně podepsané

povoleným certifikátem výrobce. Druhým z nich je, že při tomto postupu je nutný fyzický přístup k tabletu, který je vzhledem k umístění tabletu v soukromém bytě, nevhodný.

V současné konfiguraci je možné se na tablet připojit pomocí SSH s identitou roota, tudíž možnosti správy jsou dostačující. Je však potřeba za stávajících podmínek a bez fyzické demontáže tabletu z bytu zajistit možnost vzdálené aktualizace vlastním neoriginálním aktualizačním balíčkem.

4.2 Aktualizační balíčky

Aktualizační balíčky pro tento tablet jsou ve formě zip archivu, který se skládá ze dvou částí:[27]

1. META-INF - složka v archivu, která obsahuje informace o tom, jak provést aktualizaci a další metainformace
2. všechno ostatní jsou data, která mohou, ale nemusí být během aktualizace využita

Samotná instalace je posloupnost akcí popsaných v souboru */META-INF/com/google/android/updater-script*. Jedná se o skript v jazyce Edify, který obsahuje jednotlivé příkazy oddělené středníkem. V souboru *bootable/recovery/edify/README*, který je součástí AOSP je podrobnější popis syntaxe. Níže je popis příkazů, které budou použity v případě naší aktualizace:[26]

- `ui_print(msg1, ..., msgN)`
  Příkaz vypíše řetězce `msg1, ..., msgN` na displej
- `mount(fs_type, partition_type, location, mount_point)`
  Příkaz namountuje filesystém typu `fs_type` (yaffs2, ext4, ubifs) z oddílu typu `partition_type` (MTD, EMMC, UBI) a lokality `location` (system, data, cache) do místa `mount_point` ve filesystému
- `unmount(mount_point)`
  Odmountuje filesystém
- `package_extract_dir(package_path, destination_path)`
  Příkaz extrahuje složku `package_path` z archivu do místa `destination_path` ve filesystému
- `set_perm(uid, gid, mode, file1, file2, ..., fileN)`
  Příkaz nastaví uživatelské ID `uid`, skupinové ID `gid` a přístupová práva `mode` všem souborům `file1, file2, ..., fileN`

- `set_perm_recursive(uid, gid, dirmode, filemode, dir1, ..., dirN)`
  Příkaz nastaví uživatelské ID `uid`, skupinové ID `gid`, přístupová práva složky `dirmode` a přístupová práva souboru `filemode` všem souborům a složkám v složkách `dir1, ..., dirN` včetně jejich podsložek

- `symlink(target, src1, src2, ..., srcN)`
  Vytvoří symbolické linky `src1, src2, ..., srcN`, které ukazují na soubor `target`

4.3 Podepisování balíčků

Android vyžaduje, aby každá instalovaná aplikace byla digitálně podepsaná certifikátem, jehož privátní klíč vlastní vývojář aplikace. Není však potřeba, aby certifikát byl podepsaný certifikační autoritou. Běžně může být APK balíček s aplikací podepsaný tzv. self-signed certifikátem (certifikát, který je podepsaný sám sebou).[15]

Všechny soubory, které souvisejí s podepisováním balíčku jsou umístěny v jeho složce */META-INF*, a to konkrétně v souborech */META-INF/MANIFEST.MF*, */META-INF/CERT.SF* a */META-INF/CERT.RSA*.[30]

V souboru *MANIFEST.MF* jsou záznamy o SHA1 hashích jednotlivých souborů v následujícím formátu:

```
Manifest-Version: 1.0
Created-By: 1.0 (Android SignApk)

Name: system/bin/netd
SHA1-Digest: 9bnJcD4lGcEkxvMKmHZP/nI+HvM=
```

MANIFEST.MF začíná úvodními řádky, které říkají v jaké verzi soubor je a jaká aplikace byla použita pro jeho vytvoření. Následuje záznam pro každý soubor v balíčku, který udává cestu k souboru a jeho SHA1 hash kódovanou v base64.[30]

Další soubor, který vznikne podepsáním balíčku je *CERT.SF*, jehož struktura je následující:

```
Signature-Version: 1.0
Created-By: 1.0 (Android SignApk)
```

```
SHA1-Digest-Manifest: UcxGJGiLL/qiCujbpLBXhFq+Kqg=


Name: system/bin/netd
SHA1-Digest: RslUPXJSDw02FTecHpDCF2ALEQ8=
```

CERT.SF opět začíná úvodní informací o verzi a aplikaci, kterou byl vytvořen. Poté následuje SHA1 hash celého souboru *MANIFEST.MF* a následně opět dvojice název souboru a SHA1 hash pro každý soubor v balíčku. Tentokrát se ovšem nejedná o hash souboru, nýbrž o hash tří řádků z *MANIFEST.MF* týkajících se daného souboru. Stejně jako v předchozím případě jsou opět všechny hashe kódovány v base64.[30]

Pouze v souboru *CERT.RSA* je podpis samotného balíčku a zároveň je zde uchován certifikát, kterým byl balíček podepsán.[30]

Systém podepisování aktualizačních balíčků spustitelných pomocí recovery je totožný s tím, jak jsou podepisovány APK balíčky. K podpisu aktualizačního balíčku je tedy možné použít aplikaci `signapk.jar`, která je součástí AOSP a po zkompilování se nachází ve výstupní složce *out/host/linux_x86/framework*. Použití je následující:

```
$ java -jar signapk.jar -w cert.x509.pem \
> private_key.pk8 update.zip update_signed.zip
```

kde `cert.x509.pem` je certifikát, kterým má být balíček podepsán, `private_key.pk8` je privátní klíč, `update.zip` je balíček, který má být podepsán a `update_signed.zip` je výstup celé operace, tedy balíček podepsaný požadovaným certifikátem.

V případě APK nezávisí úspěch instalace na certifikátu, kterým je podepsaná, jde pouze o fakt, aby každá instalovaná aplikace byla podepsaná platným certifikátem.[15] V případě ZIP balíčku s aktualizací může být situace odlišná. Pokud má uživatel možnost instalovat ZIP balíčky podepsané jakýmkoliv certifikátem, získává možnost přepisovat jakoukoliv část operačního systému, čemuž se někteří výrobci snaží vyhnout tím, že omezí možnost instalovat ZIP balíčky z recovery pouze na ty, které byly podepsané určitým certifikátem.

V případě Ainolu jsou originální aktualizační balíčky podepisované testovacím certifikátem Googlu, jehož privátní klíč je veřejně přístupný a je možné jej stáhnout na `https://code.google.com/p/signapk/downloads/detail?name=signapk-0.3.1.tar.bz2&can=2&q=`.[14] Konkrétně se jedná

o certifikát s názvem *signapk-key.testkey.x509.pem* a k němu příslušný privátní klíč *signapk-key.testkey.pk8*. Tuto skutečnost je možné snadno ověřit na základě otisků tohoto certifikátu a certifikátu získaného ze souboru *CERT.RSA* nacházejícího se v oficiálním aktualizačním balíčku Ainolu, takto:

```
$ keytool -printcert -file CERT.RSA | grep SHA1: \
> | awk '{print $2}'
61:ED:37:7E:85:D3:86:A8:DF:EE:6B:86:4B:D8:5B:0B:FA:A5:AF:81

$ openssl x509 -sha1 -in signapk-key.testkey.x509.pem -noout\
> -fingerprint | cut -d'=' -f2
61:ED:37:7E:85:D3:86:A8:DF:EE:6B:86:4B:D8:5B:0B:FA:A5:AF:81
```

Certifikát, kterým Ainol podepisuje svoje aktualizace, je tedy stejný jako testovací certifikát Googlu, ke kterému existuje veřejně přístupný privátní klíč. Z toho plyne, že je možné vytvořit libovolnou aktualizaci, podepsat ji tímto certifikátem a následně ji nainstalovat pomocí stávajícího recovery. Tímto máme zaručeny neomezené možnosti editace softwaru v tabletu od bootloaderu přes jádro až po systémové aplikace.

4.4 Recovery systém

V případě, že je OS Android poškozen a nemůže plně nabootovat, bývá za normálních okolností nemožné se zařízením komunikovat. To je zásadní problém u zařízení, které není standardně schopné nabootovat z přípojného zařízení, protože se tím ztrácí možnost obnovit OS do funkčního stavu.

Z tohoto důvodu je součástí AOSP recovery. Jedná se o bootovatelný „u-boot legacy uImage“ uložený v zařízení na zvláštním diskovém oddílu. Při bootu Android zařízení bootloader vybere mezi dvěma oddíly, kde na jednom je uImage OS Android a na druhém uImage recovery. Výběr je ovlivněn parametry, které jsou bootloaderu buď předány při restartu zařízení z OS Android nebo jsou zadány stiskem tlačítka při startu zařízení. Z OS je možné vynutit restart do recovery příkazem `reboot recovery`. Na tabletu Ainol Novo 7 Crystal je možné vynutit nabootování do recovery držením tlačítka `Volume down` při startu tabletu.

Tablety instalované v domě obsahují běžný recovery systém, jehož zdrojové kódy jsou součástí AOSP (ve složce *bootable/recovery/*). Po nabootování nabízí recovery následující možnosti:

- `reboot system now` – restartuje systém
- `apply update from ADB` – umožňuje nahrát aktualizační balíček USB kabelem pomocí ADB (Android Debug Bridge, technologie umožňující komunikaci mezi počítačem a Android zařízením[10])
- `apply update from EXT` – umožňuje nahrát aktualizační balíček z SD karty
- `apply update from cache` – umožňuje nahrát aktualizační balíček z diskového oddílu *cache*
- `wipe data/factory reset` – smaže veškerý obsah diskových oddílů *cache* a *data* (uživatelem nainstalované aplikace a nastavení)
- `wipe cache partition` – smaže veškerý obsah diskového oddílu *cache*
- `wipe media partition` – smaže veškerý obsah diskového oddílu, který je určen pro ukládání uživatelského obsahu (oddíl, který je „vidět“ když je tablet připojen k počítači jako datové úložiště)

Jedinou možností odkud nahrát aktualizační balíček je v našem případě diskový oddíl *cache*. SD karta není v tabletech instalována a nahrávání aktualizačního balíčku přes ADB vyžaduje USB připojení, které není dostupné. Diskový oddíl *cache* je jediný oddíl, který je vždy přístupný jak OS tak i recovery a mohou si přes něj vzájemně předávat data. Ve filesystému OS je tento oddíl přimountován v */cache* s necelými 220 MB volného místa. Při běhu OS diskový oddíl standardně obsahuje dva soubory. Soubor */cache/recovery/last_install* obsahuje informaci o úspěchu/neúspěchu poslední instalace aktualizačního balíčku. V případě úspěchu soubor obsahuje 1, v případě neúspěchu 0. Soubor */cache/recovery/last_log* obsahuje log s informacemi o poslední instalaci aktualizačního balíčku.

Pro předávání informací z OS do recovery slouží soubor */cache/recovery/command*. Jeho možnosti jsou popsány v komentáři zdrojového kódu samotného recovery (*bootable/recovery/recovery.cpp*) a některé z nich jsou následující:

- --update_package=lokace – nainstaluje aktualizační balíček, který se nachází v zadané lokaci a restartuje zařízení
- --wipe_data – smaže veškerý obsah diskových oddílů *cache* a *data* a restartuje zařízení

- --wipe_cache – smaže veškerý obsah diskového oddílu *cache* a restartuje zařízení
- --just_exit – neudělá nic a restartuje zařízení

4.5 Realizace vzdálené aktualizace

4.5.1 Sestavení aktualizačního balíčku

Nejdříve je třeba sestavit aktualizační balíček, který promítne změny popsané v kapitole 3 do tabletu. Cílem aktualizace je vymazání obsahu složek */system/app* a */system/framework* a jejich naplnění novými soubory. Bohužel jazyk Edify, ve kterém je psaný aktualizační skript, nemá žádný příkaz na mazání souborů nebo složek. Obsah složky */system* je ale na samostatném diskovém oddílu, je tedy možné jej celý naformátovat a nahrát zpět všechny soubory, které se nacházely v této složce, včetně naší aktualizace.

4.5.1.1 Data v aktualizačním balíčku

Při vytváření aktualizace je možné vyjít z originálního aktualizačního balíčku, který Ainol k tomuto tabletu nabízí. Ke stažení pod názvem *Android 4.1.1.–1119.*[19] Aktualizační balíček, který je potřeba upravit, je v archivu pod názvem *g06refe2-ota-eng.grady.wang.zip*. Během aktualizace je potřeba změnit pouze obsah složky */system*, ostatní soubory v archivu lze tedy vymazat. Jedná se o složku */recovery* a soubory */logo.img*, */aml_logo.img*, */bootloader.img* a */boot.img*.

Obsah složek */system/app* a */system/framework* je nutné nahradit „deodexovanými“ původními aplikacemi s upraveným System Serverem (soubor */system/framework/services.jar*), které byly vytvořeny podle popisu v kapitole 3.

Z důvodu vzdálené správy je nezbytné zachovat možnost získat root oprávnění příkazem `su`, který se v originální aktualizaci od Ainolu nenachází. Binárku tohoto programu je možné stáhnout na `http://www.slatedroid.com/topic/39021-ainol-novo-7-crystal-root/` (soubor *Crystal_Root.zip*).[20] Z tohoto archivu je třeba zkopírovat soubory */system/xbin/su* a */system/app/Superuser.apk* do stejných složek v nově vytvořeném aktualizačním balíčku.

Vzniklý aktualizační balíček má velikost přesahující velikost volného místa na diskovém oddílu *cache*. K vyřešení tohoto problému stačí odmazat některé aplikace ze složky */system/app*, které nejsou nezbytné pro běh systému, například Google+ a Google Mapy.

4.5.1.2 Aktualizační skript

Aktualizační skript */META-INF/com/google/android/updater-script* udává, jak má recovery provést aktualizaci. Při jeho psaní jsem vyšel ze skriptu, který se nachází v originálním aktualizačním balíčku. Dále jsou popsány pouze provedené změny.

Nejdříve je nutné odstranit všechny příkazy `write_raw_image`, ty slouží k přepsání diskového oddílu obrazem. Z aktualizačního balíčku byly všechny obrazy odstraněny a v případě, že ve skriptu zůstane například: `write_raw_image(package_extract_file("bootloader.img"), "bootloader")` pokusí se recovery přepsat oddíl s bootloaderem neexistujícím obrazem, čímž dojde k smazání bootloaderu. Následkem této operace nebude tablet schopný nabootovat ani do recovery ani do OS.

V průběhu práce se mi z důvodu velkého množství provedených aktualizací (řádově stovky) podařilo dvakrát nechtěně smazat bootloader. V tomto případě naštěstí SoC umožňuje nahrát software do tabletu ve speciálním režimu a obnovit tak bootloader. Obnáší to zdlouhavou proceduru, která v krajním případě vyžaduje rozebrání tabletu a zkratování dvou pinů na čipu hlavní paměti.[21] Na obrázku 4.1 je otevřený tablet s detailem paměťového čipu. Při prvním nechtěném smazání bootloaderu bylo třeba vyzkratovat červenou čarou označené piny, aby tablet začal komunikovat alespoň se speciálním softwarem určeným pro SoC. Při druhém smazání bootloaderu tablet s tímto programem komunikoval od začátku.

Dále je možné vymazat všechny příkazy `show_progress()`, protože aktualizace se bude vykonávat vzdáleně a není potřeba na displeji informovat o jejím stádiu. Další příkazy k smazání jsou `set_bootloader_env("upgrade_step", "1")`, `package_extract_dir("recovery", "/system")`, `set_perm(0, 0, 0544, "/system/etc/install-recovery.sh")` a veškeré `assert()` příkazy.

Pro korektní fungování binárky `su` je třeba jí nastavit oprávnění `set_perm(0, 0, 6755, "/system/xbin/su")` a vytvořit jeden symbolický link `symlink("/system/xbin/su", "/system/bin/su")`. Příkazy je potřeba umístit na konec skriptu, jako poslední dva před `unmount("/system")`.

4.5.2 Vzdálená aktualizace

Nyní je nutné aktualizační balíček podepsat podle popisu v sekci 4.3, jinak bude při instalaci odmítnut recovery systémem. Toto je zásadní problém při realizaci aktualizace vzdáleně, protože při jakémkoliv selhání vypíše recovery na displej chybovou hlášku a nerestartuje tablet, čímž se ztratí

Obrázek 4.1: Rozložený tablet s detailem paměťového čipu

možnost s tabletem komunikovat a bude nutný fyzický přístup k zařízení. Dalším problémem, který může nastat je chyba v aktualizačním skriptu, je tedy nezbytné aktualizační balíček nejdříve otestovat na zařízení, které máme fyzicky přístupné a až následně jej nahrát do vzdáleného zařízení.

V tabletu není instalován program `scp`, pro přenos je možné použít pouze SSH:

```
$ ssh -p PORT root@IP cat < update.zip ">" \
> /cache/recovery/update.zip
```

kde `PORT` je port na kterém běží SSH server, `IP` je adresa tabletu a `update.zip` je cesta k aktualizačnímu balíčku. Příkaz nahraje aktualizační balíček do */cache/recovery/update.zip*. Následně je potřeba v zařízení vytvořit soubor */cache/recovery/command*, který bude obsahovat jedinou řádku:

```
--update_package=/cache/recovery/update.zip
```

Možné je to vzdáleným přihlášením na tablet a použitím programu `vi`. Následným příkazem `reboot recovery` se tablet restartuje do recovery, vykoná aktualizaci z */cache/recovery/update.zip* a restartuje se zpět do OS.

4.6 Dosažené výsledky

Kapitola popisuje postup nasazení aktualizace OS Android vytvořené v kapitole 3. Tento postup je všeobecně použitelný pro libovolný aktualizační balíček, který lze nainstalovat lokálně. Jediným omezením v tomto ohledu je velikost diskového oddílu *cache*. V případě, že aktualizační balíček přesahuje velikost oddílu je možné z něj odstranit aplikace, které nejsou nezbytně nutné pro start OS a SSH serveru a celou aktualizaci rozdělit na dvě iterace. V našem případě jsou aplikace Google+ a Google Mapy (odstraněné kvůli nedostatku místa) pro funkci zvonkového systému nepodstatné, proto není třeba je doinstalovat.

Jedním z požadavků na vzdálenou aktualizaci OS bylo zachování současného nastavení a doinstalovaných aplikací. Android je v tomto ohledu výhodně navržen, protože všechna nastavení a uživatelem doinstalované aplikace jsou na diskovém oddílu *data*, zatímco všechny soubory OS a systémové aplikace na oddílu *system*. Při naformátování oddílu *system* a jeho následném přepsání aktualizovaným OS, zůstanou uživateli všechny původní aplikace a nastavení.

Postup vzdálené aktualizace byl dlouhodobě testován v podmínkách lokální sítě. V momentě, kdy byla jeho funkčnost dostatečně ověřena v lokální

síti, byl nasazen při vzdálené aktualizaci prvního testovacího tabletu v jednom z bytů a celý proces skončil úspěšně.

Závěr

V počátcích práce bylo nutné detailně se seznámit se základy fungování OS Android na libovolném zařízení. V této fázi bylo potřeba především pochopit základní principy jako instalování aktualizací OS, získávání superuživatelských oprávnění a komunikaci se zařízením přes terminál.

V další fázi bylo třeba zaměřit se na architekturu a detailní fungování OS Android. Tato znalost je nezbytným předpokladem, jak pro analýzu možností úprav OS Android, tak i pro analýzu závady, kterou je třeba v tabletu odstranit.

Analýzou závady se podařilo zjistit, že zámek PARTIAL_WAKE_LOCK je implementován správně. Chování, kdy v okamžiku zhasnutí displeje tablet přestane komunikovat po ethernet síti, je způsobeno nevhodnou implementací systémové služby Ethernet Manager. Zpracovaná analýza možností úprav OS Android v tabletu Ainol Novo 7 Crystal byla tedy primárně zaměřena na možnosti úprav Ethernet Managera.

První způsob, jak získat neomezené možnosti úprav všech částí OS Android, je získání zdrojových kódů k požadovanému zařízení, jejich změna a následné sestavení upravené verze. Bohužel výrobce neuvolnil zdrojové kódy k tabletu Ainol Novo 7 Crystal. Společně s vedoucím práce jsme se pokusili získat zdrojové kódy alespoň od výrobce System on Chip, který je v tabletu použit, ovšem bezvýsledně. Podařilo se získat pouze zdrojové kódy k modelu Ainol Elf II, který obsahuje obdobný hardware jako Ainol Novo 7 Crystal a generické zdrojové kódy Googlu. Tyto zdrojové kódy bylo možné sestavit a spustit na emulátoru, avšak i přes veškerou snahu se nepodařilo sestavit fungující verzi pro model Crystal.

Bylo tedy nezbytné najít způsob modifikace OS Android, který lze provést i přes nedostupnost zdrojových kódů k danému modelu tabletu. První

variantou byla implementace daemona, který bude na nativní vrstvě OS spravovat ethernet rozhraní a obejde tím Ethernet Managera. Podařilo se vytvořit fungující implementaci, která urdžela ethernet rozhraní zapnuté a schopné komunikovat bez ohledu na stav displeje. Toto řešení ale mohou využít pouze aplikace běžící v nativní vrstvě OS a aplikace zajišťující funkce domovního vrátného běží v aplikační vrstvě OS. Z tohoto důvodu se toto řešení ukázalo jako nepoužitelné.

Chování Ethernet Managera se nakonec podařilo úspěšně modifikovat pomocí reverzního inženýrství. Aplikaci, obsahující implementaci Ethernet Managera, lze rozbalit na úroveň Java bytekódu, ten následně upravit, zabalit zpět a nainstalovat do tabletu. Jedna z metod Ethernet Managera zajišťuje vypínání a zapínání ethernet rozhraní v závislosti na stavu displeje. Obsah této metody je tedy možné nahradit příkazem `nop` a tím zajistit trvalé zapnutí ethernet rozhraní bez ohledu na stav displeje. Následně byl vytvořen aktualizační balíček, který aplikuje provedené změny do OS Android v tabletu.

Posledním krokem je vzdálená instalace takového aktualizačního balíčku v tabletu instalovaném v domě pomocí protokolu SSH. V průběhu práce se podařilo nalézt spolehlivý postup, jak takovou aktualizaci provést. Přiložené CD obsahuje aktualizační balíček, který odstraní nedostatek Ethernet Managera. Návod k provedení vzdálené aktualizace tabletu tímto balíčkem se nachází v příloze k této práci.

Aktualizační balíček, odstraňující vypínání ethernet rozhraní při vypnutí displeje, stejně jako proces vzdálené aktualizace OS přes SSH byl náležitě otestován. Nejdříve byly provedeny důkladné testy v podmínkách místní sítě, kterými byla simulována většina situací, které v reálném prostředí mohou nastat. Následně byla aktualizace 10. 12. 2013 nasazena k testovacímu provozu v jednom z tabletů v domě. Samotný proces vzdálené aktualizace proběhl korektně a k okamžiku vytištění této bakalářské práce nejsou pozorovány žádné chyby v chování tabletu.

Literatura

[1] Android Open Source Project: *Building Kernels [online].* [cit. 14. 12. 2013]. Dostupné z: `http://source.android.com/source/building-kernels.html`.

[2] Android Open Source Project: *Building the System [online].* [cit. 14. 12. 2013]. Dostupné z: `http://source.android.com/source/building-running.html`.

[3] Android Open Source Project: *Codenames, Tags, and Build Numbers [online].* [cit. 10. 12. 2013]. Dostupné z: `http://source.android.com/source/build-numbers.html`.

[4] Android Open Source Project: *Downloading the Source [online].* [cit. 10. 12. 2013]. Dostupné z: `http://source.android.com/source/downloading.html`.

[5] Android Open Source Project: *Initializing a Build Environment [online].* [cit. 11. 12. 2013]. Dostupné z: `http://source.android.com/source/initializing.html`.

[6] Apple Insider: *Google closes Android 3.0 Honeycomb source to prevent use on smartphones [online].* [cit. 14. 12. 2013]. Dostupné z: `http://appleinsider.com/articles/11/03/24/google_closes_android_3_0_honeycomb_source_to_prevent_use_on_smartphones`.

[7] Bhoj, V.: *Androidization of linux kernel [online].* [cit. 14. 12. 2013]. Dostupné z: `http://www.linaro.org/linaro-blog/2012/03/20/androidization-of-linux-kernel/`.

[8] dex2jar: *Guide: ModifyApkWithDexTool [online].* [cit. 17. 12. 2013]. Dostupné z: http://code.google.com/p/dex2jar/wiki/ModifyApkWithDexTool.

[9] Google Inc.: *Android architecture [online].* [cit. 16. 12. 2013]. Dostupné z: http://developer.android.com/images/system-architecture.jpg.

[10] Google Inc.: *Android Debug Bridge [online].* [cit. 14. 12. 2013]. Dostupné z: http://developer.android.com/tools/help/adb.html.

[11] Google Inc.: *Android, the world's most popular mobile platform [online].* [cit. 10. 12. 2013]. Dostupné z: http://developer.android.com/about/index.html.

[12] Google Inc.: *NotificationManager [online].* [cit. 16. 12. 2013]. Dostupné z: http://developer.android.com/reference/android/app/NotificationManager.html.

[13] Google Inc.: *PowerManager [online].* [cit. 16. 12. 2013]. Dostupné z: http://developer.android.com/reference/android/os/PowerManager.html.

[14] Google Inc.: *signapk [online].* [cit. 25. 11. 2013]. Dostupné z: https://code.google.com/p/signapk/downloads/detail?name=signapk-0.3.1.tar.bz2&can=2&q=.

[15] Google Inc.: *Signing Your Applications. [online].* [cit. 25. 11. 2013]. Dostupné z: http://developer.android.com/tools/publishing/app-signing.html.

[16] Google Inc.: *What is API Level? [online].* [cit. 10. 12. 2013]. Dostupné z: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels.

[17] Kroah-Hartman, G.: *Linux Kernel in a Nutshell.* Sebastopol: O'Reilly Media, páté vydání, 2005, ISBN 978-0-596-00930-4.

[18] Oracle: *Java SE 6 Downloads [online].* [cit. 11. 12. 2013]. Dostupné z: http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html#jdk-6u45-oth-JPR.

[19] Satomar, s.r.o.: *Ainol Novo7 Crystal - Manuál a Firmware. [online].* [cit. 25. 11. 2013]. Dostupné z: http://www.ainol-novo.cz/podpora-ainol/80-faqs/111-ainol-novo7-crystal-manual-a-firmware.

[20] Slatedroid: *Ainol Novo 7 Crystal Root [online].* [cit. 15. 12. 2013]. Dostupné z: http://www.slatedroid.com/topic/39021-ainol-novo-7-crystal-root/.

[21] Slatedroid: *Unbrick your Ainol Crystal [online].* [cit. 15. 12. 2013]. Dostupné z: http://www.slatedroid.com/topic/41478-how-to-unbrick-your-ainol-crystal/.

[22] Softpedia: *Canonical Will Remove Java From Ubuntu [online].* [cit. 11. 12. 2013]. Dostupné z: http://news.softpedia.com/news/Canonical-Will-Remove-Java-From-Ubuntu-241147.shtml.

[23] Stack Overflow: *Compiling a deodexed AOSP? [online].* [cit. 11. 12. 2013]. Dostupné z: http://stackoverflow.com/questions/8660998/compiling-a-deodexed-aosp.

[24] Stack overflow: *Guide: How to compile a C program for Android [online].* [cit. 17. 12. 2013]. Dostupné z: http://stackoverflow.com/questions/12621088/guide-how-to-compile-a-c-program-for-android.

[25] Thompson, T.: *Android Makes the Move to Power Architecture Technology [online].* [cit. 16. 12. 2013]. Dostupné z: https://community.freescale.com/community/the-embedded-beat/blog/2010/05/24/android-makes-the-move-to-power-architecture-technology.

[26] XDA Developers: *Introduction to edify updater script. [online].* [cit. 25. 11. 2013]. Dostupné z: http://forum.xda-developers.com/showthread.php?t=1669489.

[27] XDA University: *ZIP-based ROM Tweaking. [online].* [cit. 25. 11. 2013]. Dostupné z: http://xda-university.com/as-a-user/zip-based-rom-tweaking.

[28] Yadav, M.: *History of Android [online].* [cit. 25. 11. 2013]. Dostupné z: http://www.tech2crack.com/history-android/.

[29] Yaghmour, K.: *Embedded Android.* Sebastopol: O'Reilly Media, první vydání, 2013, ISBN 978-1-449-30829-2.

[30] Yang, H.: *META-INF Files - Digests, Signature and Certificate. [online]*. [cit. 25. 11. 2013]. Dostupné z: http://www.herongyang.com/Android/Project-META-INF-Files-Digest-Signature-and-Certificate.html.

[31] Zores, B.: *Dive Into Android Networking: Adding Ethernet Connectivity [online]*. [cit. 16. 12. 2013]. Dostupné z: http://elinux.org/images/9/98/Dive_Into_Android_Networking-_Adding_Ethernet_Connectivity.pdf.

Příloha A

Model nasazení

Tato příloha obsahuje popis nasazení aktualizačního balíčku odstraňujícího nedostatky Ethernet Managera na tabletu instalovaném v bytové jednotce. Aktualizační balíček opravující jeho chování se nachází v */patch/aktualizace.zip* na přiloženém CD. Ve všech následujících příkazech je třeba nahradit proměnnou `PORT` číslem portu na kterém běží SSH server v daném tabletu a proměnnou `IP` IP adresou tabletu. Při použití daemona DroidSSHd, který je v tabletech nasazen, je port implicitně nastaven na 2222. Instalaci aktualizačního balíčku *aktualizace.zip* lze provést následovně:

1. Zkopírovat aktualizační balíček do tabletu pomocí ssh:
   `ssh -p PORT root@IP cat < aktualizace.zip ">" /cache/recovery/aktualizace.zip`

2. Vytvořit v tabletu soubor */cache/recovery/command*, který řekne recovery, že má nainstalovat aktualizaci:
   `ssh -p PORT root@IP "echo --update_package=/cache/recovery/aktualizace.zip > /cache/recovery/command"`

3. Restartovat tablet do recovery:
   `ssh -p PORT root@IP "reboot recovery"`

Samotná instalace aktualizace trvá přibližně pět minut během kterých je tablet odpojen od sítě. Po dokončení je možné zkontrolovat log celé operace, který je uložen v souborech */cache/recovery/last_install* a */cache/recovery/last_log*. Jejich význam je popsán v sekci 4.4.

Po úspěšné instalaci nebude Ethernet Manager nadále vypínat ethernet rozhraní při vypnutí displeje. Nyní je možné odinstalovat program Wake

Lock, který drží trvale zámek SCREEN_DIM_WAKE_LOCK. To lze provést smazáním jeho APK balíčku ze složky */data/app*. Jméno jeho APK balíčku je závislé na způsobu instalace, ale název bude podobný tomuto: `eu.thedarken.wl-1.apk`.

Odinstalování lze v takovém případě provést následovně:

1. Smazání APK balíčku z */data/app*:
   `ssh -p PORT root@IP "rm /data/app/eu.thedarken.wl-1.apk"`
2. Restartování tabletu:
   `ssh -p PORT root@IP "reboot"`

Nyní nebude zařízení trvale udržovat zapnutý displej a zároveň nebude vypínat ethernet rozhraní při zhasnutí displeje.

Příloha B

Seznam použitých zkratek

3G Third Generation — v kontextu práce třetí generace mobilních sítí

ADB Android Debug Bridge

AOSP Android Open Source Project

API Application Programming Interface

APK Application Package File — instalační balíček OS Android aplikací

CD Compact Disc

CPU Central Processing Unit

DHCP Dynamic Host Configuration Protocol

eMMC eMultiMediaCard

ext4 Fourth Extended Filesystem

GSM Global System for Mobile Communications

HW Hardware

IPC Inter-process Communication

JDK Java Development Kit

JNI Java Network Interface

LTE Long Term Evolution — typ mobilní sítě

LTS Long-term Support

MTD Memory Technology Device

NDK Native Development Kit

OS Operační systém

RAM Random-access Memory

RPC Remote Procedure Call

SD Secure Digital

SDK Software Development Kit

SHA1 Secure Hash Algorithm 1

SoC System on a Chip

SSH Secure Shell

UBI Unsorted Block Image

ubifs Unsorted Block Image File System

USA United States of America

USB Universal Serial Bus

VM Virtual Machine

VOIP Voice over IP

XML Extensible Markup Language

yaffs2 Yet Another Flash File System 2

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `patch` ..... adresář s aktualizačními balíčky
  - `netup.zip` ..... instalace daemona netup
  - `aktualizace.zip` ..... oprava chování Ethernet Managera
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `netup` ..... řešení netup
    - `re` ..... řešení reverzním inženýrstvím
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF