Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interakcia medzi telefónom s OS Android a počítačom s OS Linux

Bakalárska práca

2015

Roman Hudec

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interakcia medzi telefónom s OS Android a počítačom s OS Linux

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: RNDr. Jaroslav Janáček, PhD.

Bratislava, 2015
Roman Hudec

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Roman Hudec
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Interakcia medzi telefónom s OS Android a počítačom s OS Linux
Interaction between an Android Smartphone and a Linux Computer

Cieľ: Cieľom práce je popísať rôzne možnosti komunikácie medzi mobilným telefónom s OS Android a osobným počítačom s OS Linux, overiť ich funkčnosť, opraviť chyby v existujúcich knižniciach a aplikáciach, prípadne doimplementovať vybranú chýbajúcu funkcionalitu. V práci budú preskúmané možnosti komunikácie pomocou USB kábla, Bluetooth a WiFi. V práci budú tiež preskúmané a demonštrované možnosti využitia rôzneho hardvéru v telefóne z prostredia OS Linux na počítači.

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 28.10.2014

Dátum schválenia: 28.10.2014 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Týmto by som sa chcel poďakovať svojmu vedúcemu RNDr. Jaroslavovi Janáčkovi, PhD. za rady, nápady a odbornú spoluprácu. Okrem toho ďakujem všetkým, ktorí vo mňa verili a podporovali ma.

Abstrakt

V tejto bakalárskej práci sme navrhli a implementovali protokol pre jednoduchý prenos súborov a senzorických dát medzi OS Android a OS Linux a systém pre prenos šifrovacieho kľúča pomocou QR kódu. Navrhnutý protokol je rozšíriteľný a podporuje šifrovanie celej komunikácie a integráciu do súborového systému pomocou knižnice FUSE.

Kľúčové slová: android, fuse, prenos súborov, dáta zo senzorov, QR kód

Abstract

We have designed and implemented a protocol for simple transfer of files and sensoric data between a Linux desktop and an Android smartphone. The protocol was designed to be extensible and to support encryption of the whole communication. We have also designed a method for transfering an encryption key using QR code and a FUSE module to better integrate into the Linux filesystem.

Keywords: android, fuse, file transfer, sensor data, qr code

Obsah

Zoznam tabuliek

Úvod

OS Android je dnes jeden z najpoužívanejších operačných systémov pre mobilné zariadenia. Čoraz viac a viac ľudí nosí svoj mobil alebo tablet so sebou na každom kroku a dnes väčšina mobilov obsahuje interné úložisko o dostatočnej veľkosti na to, aby nahradilo USB kľúč.

Lenže, ako sme zistili, stále neexistuje jednoduchý a bezpečný spôsob, ako prenášať súbory medzi OS Android a OS Linux. Tento nedostatok sa snaží riešiť táto bakalárska práca. Navyše sa zaoberáme aj tým, ako jednoducho používať sústavu senzorov v bežných Android telefónoch z Linuxových aplikácií.

V prvej kapitole sa pozrieme bližšie na bežne používané spôsoby prístupu k súborom a ich nevýhody, potom sa zamyslíme nad tým, čo očakávame od protokolu na takéto účely a navrhneme protokol, ktorý sa pokúsi splniť všetky naše očakávania. Následne ho implementujeme a na záver zhrnieme dosiahnuté výsledky práce a zamyslíme sa nad možnými vylepšeniami nášho nového protokolu a jeho referenčnej implementácie.

Kapitola 1

Súčasný stav

Interakcia medzi OS Android a OS Linux je veľmi široká téma. V tejto práci sa budeme venovať hlavne výmene dát, špecificky nás bude zaujímať najmä prenos nasledovných typov dát:

- Prístup k súborom na mobile z počítača
- Prístup k dátam z mobilných senzorov na počítači
- Ovládanie funkcionality mobilu z počítača

V tejto kapitole popíšeme ako sa dajú tieto akcie vykonávať pri použití v súčasnosti predávaných telefónov s OS Android a aktuálnych verzií software na OS Linux.

OS Android aj popisovaný software na OS Linux sú neustále vo vývoji. Preto sa situácia v jednotlivých verziách líši. Prehľad používanosti jednotlivých verzií Android-u sa dá nájsť v tabuľke 1.1

Verzia systému	Kódové označenie verzie	Verzia Android API	Podiel na trhu
2.2	Froyo	8	0.4%
2.3.3-2.3.7	Gingerbread	10	7.8%
4.0.3-4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x	Jelly Bean	17	20.3%
4.3	Jelly Bean	18	6.5%
4.4	KitKat	19	39.1%

Tabuľka 1.1: Rozdelenie verzií Android-u [12]

1.1 Prístup k súborom na mobile

V tejto sekcii popíšeme prístup k súborom na mobile prostredníctvom protokolov, ktorých podpora je zabudovaná v OS Android.

1.1.1 ADB

Android Debug Bridge je protokol určený hlavne pre vývojárov, ktorý umožňuje rôzne pokročilé možnosti prístupu k telefónu[10]. Funguje cez USB a Wi-Fi a okrem prenosu súborov podporuje nasledovné veci:

- Inštaláciu aplikácií na pripojený telefón
- Prístup k systémovému log-u cez logcat
- Zdieľanie internetu z počítača do mobilu
- Terminálový prístup k mobilu

ADB funguje na všetkých hlavných operačných systémoch vrátane OS Linux. Jeho nevýhodou je to, že musí byť v telefóne povolený, a povoľuje sa v "Developer options".

1.1.2 USB MSC

USB Mass Storage device Class je sada protokolov, určených na prístup k externému dátovému úložisku prostredníctvom USB[13]. USB MSC bol štandardne používaný v Android-e pred verziou 3.0[1]

Hlavnými nevýhodami tohoto prístupu sú:

- prostredníctvom USB MSC sa dajú zdieľať iba celé partície
- dané zariadenie môže byť naraz pripojené iba v jednom operačnom systéme.

Z prvej nevýhody vyplýva že sa nedá s počítačom zdieľať iba časť partície v internej pamäti. Ak chce výrobca umožniť zdieľanie internej pamäte počítaču prostredníctvom tohto protokolu, musí internú pamäť logicky rozdeliť na dve. Keďže sa zdieľajú celé partície, o prístup k jednotlivým partíciam a súborom na nich sa stará operačný systém na počítači. Zdieľané partície teda musia používať súborový systém, ktorý vedia používať aj všetky desktopové OS, ktoré chce výrobca podporovať, čo v praxi znamená obmedziť

[1] Táto verzia bola vydaná iba pre tablety a dnes sa už takmer vôbec nepoužíva ako vidno v tabuľke 1.1

sa na použitie FAT32, ktorý má jediný bezproblémovú podporu aj v OS Windows, OS Linux a Mac OS X [19]. Z druhej nevýhody vyplýva fakt, že aplikácie alebo dáta na zdieľanej pamäti nie sú počas zdieľania s počítačom dostupné v mobile. Teda počas toho, ako je mobil pripojený k počítaču nevie spúšťať aplikácie na zdieľanej pamäti.

Z týchto dôvodov sa tento systém používal vo väčšine prípadov iba na zdieľanie pamäťových kariet a neodporúčalo sa inštalovať na pamäťovú kartu aplikácie, ktoré vyžadujú neustály beh aj počas pripojenia k počítači (napr. aplikácie domovskej obrazovky, aplikácie na prijímanie a posielanie správ, apod.). Keď sa v mobiloch začali objavovať veľké interné pamäte (rádovo v desiatkach GB), výrobcovia ich rozdeľovali na "pamäť pre aplikácie" a "USB storage".

1.1.3 PTP

Picture Transfer Protocol je protokol, ktorý vznikol primárne na prenos fotografií z digitálnych fotoaparátov do počítača. Jeho obmedzenia sú zjavné z názvu, slúži iba na prenos fotografií, iné typy súborov sa cez neho prenášať nedajú[18].

1.1.4 MTP

Media Transfer Protocol je rozšírením protokolu PTP[14] navrhnuté pre použitie v "multimediálnych zariadeniach" ako sú napr. MP3 prehrávače, vreckové prehrávače videí, či v dnešnej dobe smartfóny. Narozdiel od protokolu PTP podporuje prenos ľubovoľných súborov a podporuje aj prenos metadát k hudobným súborom a videám. Tento protokol taktiež nemá problémy so súčasným prístupom z oboch zariadení ani nutnosť zdieľania celej partície, možno preto bol zvolený ako náhrada USB MSC v Android-e od verzie 3.0.

Vďaka MTP výrobcovia v Android-e môžu výrobcovia mať vo veľkej internej pamäti jednu partíciu, na ktorej sú aj nainštalované aplikácie aj uložené užívateľské súbory, keďže v MTP môže byť zdieľaným úložiskom aj priečinok súborov.

Hlavnými nevýhodami použitia MTP sú:

- MTP narozdiel od USB MSC podporuje iba jednu operáciu s pripojeným zariadením naraz. Nemôžem teda napr. v jednom okne kopírovať veľký súbor a v druhom prehliadať obsah priečinkov na mobile.
- MTP v jeho štandarde nemá definované operácie na editáciu súborov alebo prenos časti súboru. Ak chcem z počítača v mobile upraviť súbor, musím ho celý stiahnuť,

upraviť v počítači a celý nahrať naspäť. Toto môže byť pri väčších súboroch problematické.

- MTP je dosť voľne definovaný štandard. Zariadenie nemusí podporovať všetky príkazy definované v štandarde (ide o štandard použiteľný v rôznych typoch zariadení, dáva zmysel, že fotoaparát nemusí podporovať príkaz Prehraj

Keďže MTP je v súčasnosti preferovaný protokol na prenos súborov z mobilu do počítača, detailnejšie popíšem jeho súčasné používané implementácie pre OS Linux, ich výhody, nevýhody a funkčnosť (prípadne nefunkčnosť).

Existujú dve udržiavané a používané knižnice implementujúce komunikáciu protokolom MTP cez USB: pre Linux

- **libMTP** písaná v jazyku C. Poskytuje procedurálny interface pre volanie MTP príkazov.
- **go-mtp** písaná v jazyku Go. Poskytuje objektový prístup k MTP zariadeniam.

Obidve knižnice poskytujú taktiež funkcie pre zisťovanie funkcionality pripojených zariadení.

Existuje viacero software, ktoré používajú jednu z týchto knižníc na pripojenie obsahu MTP zariadenia ako virtuálny súborový systém. Líšia sa najmä typom použitej implementácie virtuálneho súborového systému a použitou knižnicou:

- **gvfs-mtp** Implementácia prístupu k súborom v mobile cez Gnome Virtual File System (GVFS). Poskytuje integráciu do prostredia Gnome a prostredí na ňom založených (napr. Cinnamon)). Používa knižnicu libMTP ako backend.
- **kio-mtp** Implementácia prístupu k súborom v mobile cez KDE Input/ Output (KIO). Poskytuje integráciu do prostredia KDE, používa knižnicu libMTP ako backend.
- **simple-mtpfs** Implementácia súborového systému cez FUSE (Filesystem in USErspace). K takto pripojenému zariadeniu vedia pristupovať všetky aplikácie ako k obyčajnému priečinku v súborovom systéme počítača. používa knižnicu libMTP ako backend.
- **go-mtpfs** Implementácia súborového systému cez FUSE, ktorá používa knižnicu go-mtp ako backend.

Okrem toho, že MTP je protokol, ktorý Android v súčasných verziách uprednostňuje pre prenos súborov cez USB pripojenie, existuje aj variant, v ktorom MTP prenáša súbory cez Wi-Fi. Tento variant (zvaný MTP over TCP/IP) je podporovaný iba v niektorých telefónoch s OS Android (zväčša ide o telefóny značky Sony) a na Linux- e nie je podporovaný vôbec

1.2 Prístup k dátam z mobilných senzorov a iným dátam

Android telefóny obsahujú podporu[2] pre nasledovné typy senzorov[9]:

- **Pohybové senzory**, napr. akcelerometer, gyroskop, senzor gravitácie, snímač rotačného vektora.
- **Senzory prostredia**, napr. senzory snímajúce teplotu vzduchu, tlak vzduchu, osvetlenie, prípadne vlhkosť prostredia.
- **Polohové senzory**, napr. orientačné senzory, magnetometer, lokalizácia telefónu

V súčasnosti sa cez vyššie spomínaný protokol MTP dá pristupovať k stavu batérie [14]. Zariadenie to ale musí podporovať, z dvoch testovaných zariadení to podporovalo jedno.

K dátam z iných senzorov sa jednoduchým spôsobom pristúpiť nedá. V Google play store existujú aplikácie, ktoré sú schopné v určitých intervaloch broadcastovať dáta z akcelerometra, gyroskopu alebo magnetometra, nevýhody týchto aplikácií sú väčšinou také, že každá má iný formát dát a neexistuje k nim štandardná znovupoužiteľná implementácia pre príjemcu (počítač).

[2]Nie každý telefón musí obsahovať všetky senzory

Kapitola 2

Cieľ práce a špecifikácia riešenia

2.1 Cieľ práce

Cieľom tejto bakalárskej práce je navrhnúť lepšiu sadu protokolov a nástrojov, ktorá netrpí nedostatkami súčasných možností a má dostatočne jednoduchú obsluhu aj zo strany mobilu aj zo strany počítača.

Požadované vlastnosti nami navrhovaného protokolu:

- Prenos dát bez potreby káblového spojenia medzi mobilom a počítačom
- Pripojenie na jedno "kliknutie" ak je mobil pripojený k rovnakej sieti ako počítač
- Bezpečné šifrované spojenie
- Možnosť pripojenia len k dôveryhodným zariadeniam a rozumný spôsob spárovania zariadení
- Možnosť práce s priečinkami počas prenosu veľkého súboru
- Rozšíriteľnosť protokolu
- Možnosť inštalácie na zariadenia bez podpory spúšťania aplikácií s právami administrátora (root)
- Možnosť prístupu k súborom v mobile ako keby sa nachádzali na disku počítača (Integrácia do súborového systému)
- Možnosť programátorského prístupu k súborom a senzorickým dátam

2.2 Špecifikácia riešenia

2.2.1 Komunikácia

Ako vhodný komunikačný kanál sa zdá byť použitie Wi-Fi, keďže v dobe písania tejto práce vie dosiahnuť väčšie rýchlosti ako Bluetooth, nevyžaduje káble, a väčšina telefónov s OS Android je schopné vytvoriť Wi-Fi sieť, na ktorú sa vie počítač v prípade neexistencie inej siete pripojiť.

Keďže táto práca sa zaoberá prístupu k súborom, iným dátam a funkcionalite mobilu z počítača, protokol sme navrhli podľa modelu klient-server, kde telefón s OS Android plní rolu servera, ktorý odpovedá na požiadavky počítača = klienta.

2.2.2 Integrácia do systémov OS Android a OS Linux

Android

Aby bolo naše riešenie čo najlepšie prístupné pre bežného používateľa. Rozhodli sme sa, že Android-ový komunikačný server bude vo forme obyčajnej Android aplikácie. To umožní publikovať našu aplikáciu do Google Play Store, vďaka čomu si ju budú môcť používatelia jednoducho na pár kliknutí nainštalovať.

Linux

Aby bolo naše riešenie čo najlepšie integrované do systému OS Linux, rozhodli sme sa, že ukážkový komunikačný klient bude implementovaný s pomocou knižnice FUSE[1], ktorá umožňuje pripojiť si súborový systém Androidu priamo do súborového systému na Linuxe.

2.2.3 Rozšíriteľnosť protokolu

Informačné technológie sa vyvíjajú veľmi rýchlo a preto aby sa software presadil, musí byť istým spôsobom nadčasový a pripravený na budúcnosť. S možnosťou budúceho rozšírenia komunikačných protokolov a možností nášho software je najlepšie počítač už pri návrhu.

Návrh riešenia

Okrem príkazov definovaných v tejto špecifikácii budú môcť existovať rozšírenia obsahujúce ďalšie príkazy, ktoré nie sú vopred špecifikované. Medzi povinne implementované

[1] Filesystem in UserSpace

príkazy bude zaradený príkaz EXTENSIONS, na ktorý server bude odpovedať zoznamom všetkých ním podporovaných rozšírení. Takýmto spôsobom si klient, ktorý chce od servera niečo navyše vedieť zistiť, či ním požadované rozšírenie server podporuje.

2.2.4 Bezpečné šifrované spojenie a dôveryhodnosť druhej strany

Keďže mobil je v dnešnej dobe častokrát používaný ako pracovný nástroj, na ktorom môžme prenášať súbory, ku ktorým nechceme dávať ostatným ľuďom prístup, naskytá sa požiadavka na bezpečné šifrované spojenie medzi mobilom a počítačom. Taktiež chceme zaručiť, že zariadenie, s ktorým komunikujeme je skutočne to, s ktorým si myslíme, že komunikujeme. Od zvoleného šifrovacieho systému by tiež bolo vhodné očakávať, aby nemala príliš veľký vplyv na rýchlosť prenosu súborov.

Návrh riešenia

Ako vhodný šifrovací algoritmus sme vybrali AES-CBC s 128-bitovými kľúčmi. Na túto šifru v súčasnosti neexistuje výpočtovo prijateľný útok, takže môže byť považovaná za bezpečnú. Je štandardizovaná, existuje k nej implementácia pre Javu v rámci JCE (Java Cryptography Extensions), s ktorou je Android kompatibilný a taktiež implementácia pre jazyk C++, ako súčasť balíka OpenSSL. Je tiež dostatočne rýchla na to, aby nespomaľovala prenos súborov na dnešných Wi-Fi sieťach.

Keďže sme sa rozhodli, že protokol bude nezávislý od komunikačného kanálu, musia od neho byť nezávislé aj identifikátory jednotlivých zariadení. Ako zvolený typ sme vybrali štandardizovaný identifikátor typu UUID verzie 4 (náhodne generované UUID). Pravdepodobnosť, že dve zariadenia budú mať rovnaké UUID je veľmi malá (UUID obsahuje 122 náhodných bitov) a UUID sú prenositeľné medzi zariadeniami, v prípade potreby pregenerovateľné a na Linux-e nie sú viazané na desktop ale na jeho používateľa (na Android-e je UUID zariadenia uložené v konfigurácii aplikácie, na Linux-e sú uložené v konfiguračnom súbore v používateľom domovskom priečinku).

2.2.5 Vyhľadanie mobilu, ak sa nachádza v blízkosti počítača

Na Wi-Fi sieti sa okrem mobilu a počítača, ktoré chceme spojiť môžu nachádzať aj stovky rôznych ďalších zariadení. Je vhodné, aby bol každý mobil na sieti nejakým spôsobom odlíšiteľný od ostatných, ideálne bez nutnosti pamätať si jeho UUID. Pri prvom spojení sa tiež zíde, ak existuje protokol pre nájdenie kompatibilných zariadení na sieti.

Návrh riešenia

Rozhodli sme sa, že počas toho, ako bude mobil schopný prijať spojenie bude ohlasovať svoje UUID pomocou UDP broadcast-u na vopred špecifikovanom porte 9247 každú sekundu. Na strane Linux-u môže potom existovať program, ktorý zachytáva UDP broadcast-y na tomto porte, a pre každý mobil zobrazí jeho IP adresu, UUID a informáciu, či je mobil spárovaný s daným používateľom počítača, aby používateľ vedel veľmi ľahko odlíšiť svoj mobil od ostatných mobilov na danej WiFi sieti používajúcich rovnaký protokol.

2.2.6 Programátorský prístup k možnostiam protokolu

Linuxová filozofia kladie dôraz na rozšíriteľnosť kódu. Bolo by veľmi užitočné, keby sa komunikácia navrhovaným protokolom dala jednoducho implementovať do nových nástrojov, naprogramovaných inými programátormi.

Návrh riešenia

Rozhodli sme sa, že na strane Linux-u budú všetky protokolové funkcie vyčlenené do zdieľanej C++ knižnice libalc.so a knižnica, spolu so zdrojovými kódmi bude distribuovaná ako open source. K práci bude tiež pripojená ukážka použitia protokolu pomocou tejto knižnice bez pripájania FUSE súborového systému.

Kapitola 3

FUSE

Filesystem in Userspace (FUSE) je modul jadra pre Unixové operačné systémy, ktorý umožňuje programátorom implementovať ich vlastný súborový systém bez zásahu do jadra tak, že s ním môžu pracovať aj neprivilegovaní používatelia. Toto je dosiahnuté tak, že samotný súborový systém je spustený s právami bežného používateľa a knižnica FUSE prekladá systémové volania na volania metód implementovaných používateľom.

Pomocou FUSE je napr. implementovaný ovládač pre prácu so súborovým systémom NTFS pre Linux zvaný ntfs-3g, vyššie spomínaný mtpfs, alebo linuxová verzia TrueCryptu.

3.1 Výber programovacieho jazyka

Autori FUSE poskytujú programovacie rozhranie pre FUSE súborové systémy v jazyku C. Existujú však aj neoficiálne open source API pre rôzne ďalšie programovacie jazyky, ako napr. Javu, Python, Haskell alebo Perl.

3.2 Inicializácia FUSE systému

Program inicializujúci súborový systém musí vo svojej main funkcii zavolať funkciu fuse_main, ktorej zavolaním odovzdá knižnici libfuse smerníky na implementácie jednotlivých volaní súborového systému, smerník na ľubovoľne definovanú štruktúru na súkromné dáta súborového systému, kde si súborový systém vie uložiť napr. vlastné informácie o aktuálne otvorených súboroch, cestu k priečinku, do ktorého má byť súborový systém pripojený a parametre, s ktorými sa má FUSE systém zavolať.

Pomocou parametrov sa dá napríklad nastaviť, či súborový systém podporuje vykonávanie viacerých operácií naraz, najväčšia veľkosť naraz čítaného (alebo zapisovaného) bloku, automatický preklad kódovania súborových mien, meno súborového systému (ak sa explicitne nenastaví, použije sa meno programu, ktorý volá fuse_main),

Po zavolaní funkcie fuse_main môže main funkcia volajúceho programu skončiť. Volaním fuse_main, sa totiž vytvorí nové vlákno, v ktorom beží riadiaci cyklus FUSE systému, ktorý dostáva príkazy od modulu v jadre.[16]

3.3 Odpojenie FUSE systému

Odpojiť FUSE súborový systém je možné štandardne zavolaním linuxového príkazu umount, ak má používateľ práva ho použiť, prípadne FUSE príkazom fusermount -u, ktorý by mal byť vždy použiteľný používateľom, ktorý súborový systém pripojil.

3.4 Implementácia metód súborového systému v jazyku C++

Keď používateľ pristupuje k jednotlivým prvkom súborového systému (priečinkom, súborom, odkazom, ...), jeho akcie sú prekladané na Unixové systémové volania (napr. open, read, write, stat), ktoré knižnica FUSE prekladá na volania metód implementovaných naším súborovým systémom. Náš súborový systém nemusí implementovať všetky z týchto metód, niektoré sú nahraditeľné inými (napr. namiesto volania create() sa dá volať mknod() a open()), a knižnica FUSE volí vhodné náhrady, pokiaľ náhrada za neimplementované volanie existuje. Niektoré volania sa pri bežnej práci (kopírovanie súborov) nepoužívajú (napr. fallocate()).

Väčšina týchto metód má rovnakú hlavičku ako k nim prislúchajúce systémové volanie. Niektoré však tvoria výnimku a umožňujú svoju funkcionalitu vykonať viacerými spôsobmi (napr. readdir()). Od systémových volaní súborového systému sa líšia hlavne tým, že kód chyby sa nevracia v premennej errno, ale ako návratová hodnota jednotlivých funkcií.

Podrobnejší popis jednotlivých metód, ich hlavičiek, vrátane toho, ako sme ich implementovali sa dá nájsť v kapitole o implementačných detailoch Linux-ového kódu.

Kapitola 4

OS Android

Android je operačný systém, ktorý používa jadro Linux. Avšak v userspace sa od bežných linuxových distribúcií pre počítače líši.

Userspace bežných linuxových distribúcií sa väčšinou skladá zo sady terminálových nástrojov, grafického prostredia založenom na správcovi okien a správcu balíčkov, pomocou ktorého sa spravujú programy, ktoré sa inštalujú do Unix-ovej súborovej štruktúry.

V android-e sa nekladie dôraz na terminálové nástroje, takže sú dostupné len tie najzákladnejšie. Keďže Android je systém, ktorý bol od základu navrhnutý na to, aby bežal na zariadeniach s malými obrazovkami (telefónoch) a teda nebola požadovaná funkcionalita behu viacerých programov vedľa seba, Android-u chýba akýkoľvek správca okien, nepoužíva teda X Window system.

V Androide sú aplikácie balené do balíčkov formátu apk, ktorý je založený na formáte jar, každá aplikácia je v rámci androidovej súborovej štruktúry inštalovaná do vlastného zabezpečeného priečinka a počas svojho behu vie pristupovať iba ku svojim súborom, pokiaľ si pri inštalácii alebo pri svojom behu nevypýta práva na prístup k iným zložkám. Prístup k systémovým súborom, alebo k súborom iných aplikácií štandardne nie je získateľný.

Android poskytuje množstvo API funkcií na programovanie aplikácií pre systém Android. Toto API je v dobe písania tejto bakalárskej práce dostupné iba pre jazyk Java. Je však možné implementovať časť funkcionality v jazyku C++ spolu s prístupom k niektorým natívnym Linuxovým API pomocou NDK (Native Development Kit).

4.1 Zloženie Android aplikácie

V Android-e sa aplikácia skladá z Manifest-u a z komponent.

4.1.1 Manifest

Manifest je XML dokument, v ktorom je definované z akých komponent sa aplikácia skladá a aké povolenia od používateľa bude používať. Tieto povolenia sú následne od používateľa vypýtané už pri inštalácii, v súčasnej verzii Android-u (5.1) nie je možnosť niektoré povolenia selektívne nepovoliť (aj keď od verzie 5.0 je možné zakázať niektorým aplikáciam prístup k internetu pokiaľ nie je k dispozícii WiFi pripojenie). Pokiaľ nejaká aplikácia od novej verzie vyžaduje viac povolení, musí byť aktualizovaná manuálne a používateľ musí nové povolenia potvrdiť.

4.1.2 Komponenty

V systéme Android existujú tieto 4 typy aplikačných komponent.[6]

Aktivita

Jedna aktivita reprezentuje jednu obrazovku s používateľským rozhraním, ktorá je zobrazená používateľovi. Napr. aplikácia email-u môže mať jednu aktivitu pre zobrazenie zoznamu emailov, ďalšiu aktivitu pre zobrazenie prijatého emailu a ďalšiu aktivitu v ktorej môže používateľ napísať a odoslať nový mail. Každá aktivita je nezávislá od ostatných a môže byť spustená z externej aplikácie (napr. v aplikácii fotoaparátu môže byť tlačítko na poslanie fotky mailom).

Služba

Služba je komponenta, ktorá beží v pozadí a nemá používateľské rozhranie. Iné komponenty, napr. aktivity môžu služby spúšťať a interagovať s nimi. Služba môže bežať aj keď je aplikácia v pozadí.

Content provider

Content provider spravuje dáta zdieľateľné medzi jednotlivými aplikáciami. Ostatné aplikácie môžu pomocou Content provider-ov čítať prípadne meniť zvolenú sadu dát z aplikácie, v ktorej sa daný content provider nachádza. Napr. systém Android poskytuje content provider, ktorý spravuje používateľove kontakty. Ak má nejaká aplikácia povolenie pristupovať ku kontaktom, pristupuje k nim práve pomocou prideleného content providera.

Broadcast receiver

Broadcast receiver je komponenta, ktorá je schopná reagovať na upozornenia od systému alebo iných aplikácií. Pomocou broadcast receiverov sa dá napr. reagovať na zhasnutie obrazovky, vybíjajúcu sa batériu, odfotenie obrázku a podobne. Broadcast receiver nemá používateľské rozhranie ako také, ale môže vytvoriť notifikáciu, alebo zobraziť správu v malom pop-up-e.

4.2 Životný cyklus Android aplikácie

V manifeste má aplikácia definovanú nanajvýš jednu aktivitu, ktorá je zobrazená v menu aplikácií. (Môžu existovať aplikácie, ktoré žiadne v menu zobrazené aktivity nemajú a slúžia iba na to, aby poskytovali funkcionalitu iným aplikáciam)[6]

Iné aktivity alebo služby môžu byť spustené tým, že aplikácia pošle systému Intent s vhodne nastavenými parametrami. Intent (anglicky úmysel) je v OS Android objekt, pomocou ktorého môžu jednotlivé aplikačné komponenty medzi sebou interagovať.[6]

Životný cyklus aplikácie v OS Android je kompletne manažovaný operačným systémom. Bez dodatočného nastavenia všetky komponenty jednej Android aplikácie bežia v rámci jedného procesu, avšak v manifeste sa toto správanie dá zmeniť. Z hľadiska OS Android existuje 5 typov procesov (v poradí podľa dôležitosti)[8]

- **Proces v popredí**: Proces, v ktorom beží práve aktívna aktivita, metóda onReceive v BroadcastReceiveri, alebo štartujúca služba
- **Viditeľný proces**: Proces, v ktorom beží aktivita, ktorá je sčasti viditeľná, ale nie je práve aktívna, napr. keď je prekrytá inou aktivitou zobrazenou v dialógu
- **Proces služby**: Proces, v ktorom beží aktívna služba
- **Proces v pozadí**: Proces, v ktorom beží Aktivita, ktorá nie je používateľom viditeľná (napr. používateľ z nej vyšiel a prešiel do inej aplikácie) Takéto procesy sú v prípade nedostatku operačnej pamäte zabíjané v závislosti od toho, kedy boli naposledy viditeľné
- **Prázdny proces**: Proces, v ktorom aktuálne nebeží žiadna aplikačná komponenta.

Operačný systém v prípade nedostatku pamäte zabíja procesy podľa dôležitosti daného procesu.

Pri zmene stavu je na danej aktivite systémom zavolaná príslušná metóda (napr. pri prechode medzi stavom v popredí a viditeľnou je na aktivite zavolaná metóda onPause()).

4.3 NDK

Native Development Kit (NDK) je sada systémových knižníc a kompilátorov umožňujúcich programovať niektoré časti android aplikácií v jazyku C a kompilovať ich do natívneho kódu. Je založená na cross-compile toolchain-och a Java Native Interface (JNI)

4.3.1 Podpora kompilátorov a CPU architektúr

NDK je sada cross-compile toolchainov, nástrojov schopných kompilovať kód určený pre inú architektúru CPU ako tá, na ktorej je samotný nástroj spustený. V aktuálnej verzii NDK (v dobe písania bakalárskej práce) sú obsiahnuté kompilátory GCC vo verziách 4.8 a 4.9 a Clang vo verziách 3.5 a 3.6, s podporou kompilovania pre architektúry ARM od Android verzie 1.0, x86 a MIPS od Android verzie 2.3 a architektúry ARM64, x86_64 a MIPS64 od Android verzie 5.0.

4.3.2 Podpora knižníc

Na všetkých architektúrach sú stabilne podporované knižnice libc (štandardné knižnice jazyka C), libm, všetky funkcie z JNI, libz, liblog, OpenGL ES 1.1 a 2.0, libjnigraphics, minimálna podpora C++, OpenSL ES, Niektoré natívne Android API.[11]

NDK sa dá v Android-e využiť dvomi spôsobmi.

- Naprogramovať niektoré funkcie v rámci triedy natívne.
- Naprogramovať natívne celú aktivitu. Natívna aktivita umožňuje procedurálne naprogramovať OpenGL ES aplikáciu bežiacu na Android-e bez akéhokoľvek Java kódu. Natívne aktivity sú podporované od Android verzie 2.3

Pomocou NDK nie je možné natívne implementovať celé služby alebo Content Providerov.

4.3.3 Implementácia natívnej funkcie

Na to, aby bola z Java kódu zavolateľná funkcia naprogramovaná v jazyku C alebo C++, musí k nej existovať v nejakej triede metóda s kľúčovým slovom native a žiadnou

Java implementáciou a v tejto triede musí byť staticky načítaná skompilovaná knižnica s implementáciou danej metódy pomocou volania System.loadLibrary().

Ako argument pre System.loadLibrary() je použité platformovo nezávislé meno knižnice, v Android-e je to meno skompilovaného súboru bez prefixu lib a prípony .so, keby sme pomocou JNI implementovali natívne knižnice pre Java aplikácie pod Windows-om, bolo by to meno knižnice bez prípony .dll.

V danej knižnici sa potom musí nachádzať implementácia metódy s názvom odvodeným od celého mena triedy v ktorej sa nachádza a metódy ktorú implementuje. Napr. ak ideme natívne implementovať metódu nativeFunction v triede com.example.native, musíme v knižnici mať funkciu jazyka C s názvom Java_com_example_native_nativeFunction.

Typy argumentov a návratovej hodnoty musia byť kompatibilné s príslušnými typmi prislúchajúcej metódy v jazyku Java. JNI poskytuje definície typov prislúchajúce k štandardným typom jazyka Java. Pozrieť si ich môžete v tabuľke 4.1.

4.4 Povolenia aplikácií

Z bezpečnostných dôvodov v systéme Android aplikácie štandardne nemajú prístup k žiadnym operáciam, ktoré by mohli nejakým spôsobom ovplyvniť správanie systému alebo iných aplikácií, ani pristupovať k súkromným dátam používateľa.

Na to, aby aplikácia vedela použiť niektorú z chránených funkcií, musí mať v manifeste deklarované príslušné povolenia pomocou tagov <uses-permissions>.[7]

Povolenia sú aplikácii udelené pri inštalácii. Nainštalovaná aplikácia má po svojom spustení automaticky prístup ku všetkým povoleniam, ktoré si pri inštalácii vypýtala a nemôže si vypýtať počas svojho behu žiadne ďalšie.

Ak sa aplikácia pokúsi zavolať funkciu, ku ktorej nemá povolenie, väčšinou je vyhodená výnimka typu SecurityException. Avšak v dobe písania tejto práce by nemala nastať situácia, že aplikácia sa nainštaluje s tým, že nedostane všetky práva, ktoré si pri inštalácii vypýtala, takže manuálne ošetrovanie týchto výnimiek v kóde nie je nutné.

boolean	jboolean
byte	jbyte
char	jchar
short	jshort
int	jint
long	jlong
float	jfloat
double	jdouble
void	void
Object	jobject
Class	jclass
String	jstring
[]	jarray
object[]	jobjectArray
boolean[]	jbooleanArray
byte[]	jbyteArray
char[]	jcharArray
short[]	jshortArray
int[]	jintArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray
Throwable	jthrowable

Tabuľka 4.1: Podporované dátové typy v JNI a k nim prislúchajúce primitívne typy resp. triedy v jazyku Java[17]

4.4.1 Prístup k súborom na internom úložisku telefónu a vymeniteľnej SD karte

Operačný systém Android začal od verzie 4.4 prerábať prístupové práva k úložným zariadeniam. Vo väčšine telefónov sa už od verzie 4.0 interné úložisko správalo ako emulovaná SD karta. Umiestnenie externej SD karty v súborovom strome nebolo do verzie 4.4 presne definované a neexistovali API pre jej nájdenie. Do verzie 4.4 existovali 2 povolenia: READ_EXTERNAL_STORAGE a WRITE_EXTERNAL_STORAGE, ktoré umožňovali prístup ku všetkým SD kartám (emulovaným aj reálnym) na čítanie resp. čítanie aj zapisovanie.

Od verzie 4.4 v systéme Android existuje jedno primárne úložné zariadenie a zvyšné úložné zariadenia sú klasifikované ako sekundárne. Pribudli nové API pre vytváranie a hľadanie aplikačných zložiek na sekundárnych úložných zariadeniach a zmenili sa štandardné prístupové práva pre sekundárne úložné zariadenia. Od Android 4.4 sa povolenie WRITE_EXTERNAL_STORAGE týka len primárneho úložného zariadenia (v prípade telefónov s interným úložným priestorom ide paradoxne o tento priestor). Bez akéhokoľvek povolenia majú aplikácie právo zapisovať do vlastného priečinka na sekundárnych úložiskách a s povolením READ_EXTERNAL_STORAGE môžu čítať súbory iných aplikácií na sekundárnych úložiskách. To znamená, že v operačnom systéme Android 4.4 aplikácia štandardne nemôže získať prístup na zapisovanie do priečinkov a súborov na sekundárnom úložisku, ktoré sú mimo jej priečinka, teda napr. aplikácia nemôže získať prístup k mazaniu fotografií uložených na SD karte alebo editácii ID3 tagov v hudobných súboroch uložených na SD karte.[1]

Od verzie Android 5 existuje nové API pre prístup k súborom s pomocou operačného systému. Funguje tak, že aplikácia si v ľubovoľnom momente môže od užívateľa vypýtať prístup k vybranej zložke (vrátane jej podzložiek), používateľ môže následne takto povolený prístup aplikácii odobrať. Ak má aplikácia takto vypýtaný prístup k nejakej zložke, vie poprosiť systém o funkcinoalitu, ktorá vyžaduje práva na zápis, tj. vytvorenie priečinka, vytvorenie súboru, prípadne otvorenie súboru na zápis prostredníctvom unixového file descriptoru, prípadne Java objekt typu OutputStream na zapisovanie do daného súboru.[2]

V starších verziách Android-u 4.4 existovali techniky, ktorými sa dalo aj napriek týmto zmenám zapisovať do súborov na sd karte, vďaka zneužívaniu chýb v implementácii API pre prístup k súborom pomocou OS. Takéto metódy používalo zopár správcov súborov pre OS Android[3]. V novších verziách OS Android už boli príslušné chyby opravené. Okrem toho pre zariadenia, na ktorých sa dá zapisovať do systémových súborov[1] sa dá zmeniť správanie povolenia WRITE_EXTERNAL_STORAGE aby všetky aplikácie ktoré ho majú mali prístup k zapisovaniu na všetky súbory na všetkých nesystémových úložiskách.

[1]Existujú rôzne exploit-y, pomocou ktorých sa dá na zariadenie dostať setuid aplikácia su a pomocou nej sa dajú získať práva používateľa root vrátane zápisu do systémových súborov

Kapitola 5

Návrh protokolu

5.1 Štruktúra navrhnutého protokolu

Protokol bol navrhnutý tak, aby mohol byť postavený nad ľubovoľným spoľahlivým[1] protokolom. Takýmto spôsobom je možné jednoduchšie inicializovať FUSE súborový systém na strane Linux-u (funkcia main() nečaká na prichádzajúce spojenie) a mať istotu, že dlhšie odpovede, ktoré sú pri niektorých príkazoch potrebné (napr. pri príkaze readdir()) prídu v celosti a v správnom poradí. Protokol bol pomenovaný Android-Linux Communication Protocol (ALC).

Kvôli jednoduchosti spracúvania príkazov a definície štruktúry požiadaviek a odpovedí sme sa rozhodli pre použitie textovo založeného protokolu. Protokol bol navrhnutý jednostranne, aby sme vedeli jednoducho priradiť požiadavku k odpovedi. Požiadavky môžu byť posielané iba zo strany OS Linux.

Keďže vo väčšine príkazov súborového systému môže nastať nejaká chyba pri spracúvaní (napr. chýbajúce prístupové oprávnenia, snaha čítať neexistujúci súbor, apod.), rozhodli sme sa, že odpoveď na každú požiadavku bude obsahovať na prvom riadku buď 0 ak sa požiadavke podarilo vyhovieť, alebo kód chyby. Požiadavka môže byť zložená z príkazu a argumentov oddelených medzerou na jednom riadku, odpoveď môže byť podľa potreby rozdelená na viacero riadkov (napr. v príkaze readdir(), keďže v názve súboru sa v bežne používaných súborových systémoch môže nachádzať medzera, ale nie znak konca riadku). Ukážky komunikácie navrhnutým protokolom sú k nájdeniu pri popise jednotlivých operácií, ktoré protokol podporuje.

[1] Spoľahlivý protokol zaručuje, že všetky poslané správy dojdú v celosti a v správnom poradí

V prípade, že treba preniesť nejaké dáta, ktoré môžu obsahovať znak konca riadka, dáta sa zakódujú do formátu Base64 a pošlú sa na jednom riadku. Takto sa napr. kódujú prílohy pri posielaní emailov, kde sa tiež používa textový protokol.

5.2 Vytvorenie a ukončenie spojenia

V štandardnej implementácii, ktorá je priložená k tejto práci, sme sa rozhodli náš protokol postaviť nad protokol TCP. Ako sieť sme sa rozhodli použiť WiFi, keďže v dnešnej dobe je WiFi sieť bežnou súčasťou pracoviska a v prípade nedostatku WiFi siete väčšina mobilov podporuje vytvorenie WiFi hotspotu, na ktorý sa následne počítač vie pripojiť.

Android telefóny sú schopné vypínať WiFi počas nečinnosti z dôvodu šetrenia batérie. Aby sme nenarúšali túto a podobné šetriace funkcie, navrhli sme, že na pripojenie k mobilu bude potrebné na mobile explicitne zapnúť komunikačný server. Na komunikačný server sa môže následne napojiť ľubovoľné množstvo zariadení (Pre každé zariadenie sa vytvorí samostatné komunikačné vlákno). Komunikačný server je potom kedykoľvek vypnuteľný z notifikačnej lišty telefónu, čím sa ukončia všetky prebiehajúce spojenia.

5.3 Párovanie zariadení

Pri párovaní potrebujeme zaistiť, že sa vygeneruje bezpečný náhodný šifrovací kľúč na jednom zariadení a bezpečným spôsobom sa prenesie na druhé zariadenie. Chceme mať šifrovací kľúč viazaný na dvojicu zariadení, medzi ktorými bude šifrovať všetku komunikáciu, takže potrebujeme vedieť k šifrovaciemu kľúču jednoznačne priradiť zariadenie, ku ktorému patrí. Šifrovacie kľúče a UUID zariadení sú uložené v konfigurácii knižnice protokolu, na Android-e v štandardnom úložisku pre konfiguráciu, v Linux-e v priečinku /.config/alc/. Protokol nešpecifikuje spôsob, akým sa zariadenia budú párovať.

V štandardnej implementácii párovanie prebieha tak, že Linuxový klient sa pripojí na zariadenie, s ktorým sa chceme párovať, vypýta si jeho UUID, vygeneruje šifrovací kľúč, uloží si ho a vygeneruje QR kód, ktorý obsahuje UUID linuxového klienta a daný šifrovací kľúč. Android následne dostane notifikáciu, že sa s ním linuxový klient chce spárovať, po kliknutí na túto notifikáciu sa používateľ dostane do hlavného menu Androidovej aplikácie, kde prostredníctvom možnosti pair môže otvoriť skener QR kódov a nasnímať QR kód vygenerovaný na obrazovke počítača. Z QR kódu sa následne

prečíta UUID a šifrovací kľúč a uloží sa do konfigurácie. Potom sa už môže Linuxový klient bezpečne pripojiť bez nutnosti reštartovania serveru na Androide.

5.4 Zapnutie/vypnutie šifrovania

Dve už spárované zariadenia môžu po nadviazaní spojenia začať šifrovať celé spojenie. Aby mohlo byť šifrovanie hocikedy zapnuteľné a neovyplyvnilo to veľmi štruktúru kódu, šifrujeme dáta po posielaných riadkoch, a zašifrovaný riadok je zakódovaný kódovaním špecifikovaným v šifrovacom algoritme. Protokol má pre zapnutie šifrovanie operáciu ENCRYPT s dvomi argumentom: UUID zariadenia na druhej strane spojenia a šifrovacím algoritmom. V referenčnej implementácii je podporovaný jediný algoritmus AES-128-CBC-B64 (šifra AES-128 v móde CBC a výsledný šifrový text zakódovaný do Base64). Ďalšie metódy šifrovania môžu byť doimplementované v budúcnosti ako rozšírenie protokolu. Voľba metódy šifrovania je iba u klienta = na strane OS Linux.

5.5 Základné operácie protokolu

Protokol bol navrhnutý modulárne. Štandardnou súčasťou protokolu je len malá skupina operácií, zvyšné sú prístupné ako rozšírenia protokolu. Medzi základné operácie protokolu patria:

- **VERSION** Operácia nemá žiadne argumenty, odpoveďou je reťazec s označením verzie protokolu, ktorú server používa. V štandardnej implementácii na strane servera je odpoveďou reťazec BETA1.
- **IDENTIFY** Operácia nemá žiadne argumenty, odpoveďou je reťazec s UUID servera.
- **PAIR** Argumentom je UUID klienta. Na túto operáciu server neodpovedá, namiesto toho pošle systému Android notifikáciu o požiadavku na párovanie.
- **CANENCRYPT** Argumentom je UUID klienta. Prvým riadkom odpovede je reťazec YES alebo reťazec NO, podľa toho, či na strane servera existuje šifrovací kľúč určený pre tohto klienta. Druhým riadkom odpovede je UUID servera.
- **ENCRYPT** Argumentom sú UUID klienta a šifrovací algoritmus. Prvým riadkom odpovede je reťazec OK alebo FAIL, podľa toho, či na strane servera existuje šifrovací kľúč určený pre tohto klienta a či server podporuje zvolený šifrovací algoritmus. Druhým riadkom odpovede je nešifrované UUID servera.

- **CRYPTHELLO** Táto operácia slúži na otestovanie, či obidve zariadenia používajú rovnaký šifrovací kľúč. Argumentom je ľubovoľný reťazec. Odpoveďou od servera by mal byť ten istý reťazec.

- **EXTENSIONS**. Operácia nemá žiadne argumenty. Prvým riadkom odpovede je číslo n udávajúce počet rozšírení podporovaných sererom. Ďalej nasleduje n riadkov, pre každé rozšírenie jeho identifikátor.

5.6 Rozšírenie pre prácu so súborovým systémom

Prvým rozšírením, ktoré sa nachádza v štandardnej implementácii priloženej k tejto bakalárskej práci, je rozšírenie pre pripojenie súborového podstromu v telefóne s OS Android ako FUSE súborový systém k počítaču s OS Linux. Rozšírenie dostalo v protokole identifikátor FUSE-FS-1, kde číslo 1 označuje verziu tohto rozšírenia. Súčasťou tohto rozšírenia protokolu sú nasledovné operácie:

- **PREFIX** Argumentom je cesta v Android súborovom systéme, ktorá bude použitá ako koreň pripojeného súborového podstromu. Odpoveďou je reťazec OK alebo FAIL, podľa toho, či daná cesta ukazuje na existujúci priečinok, alebo nie.

- **GETATTR** Argumentom je cesta k súboru. Prvým riadkom odpovede je výsledok operácie (0 ak bola operácia vykonaná úspešne, alebo kód chyby vypísateľný pomocou funkcie perror ak operácia skončila chybou). Ďalej nasleduje 13 riadkov reprezentujúcich jednotlivé položky v linuxovej štruktúre stat. Ak operácia skončila chybou, tieto čísla môžu byť náhodné.

- **STATFS** Argumentom je cesta k priečinku, Prvým riadkom odpovede je výsledok operácie (0 ak bola operácia vykonaná úspešne, alebo kód chyby). Ďalej nasleduje 11 riadkov reprezentujúcich položky linuxovej štruktúry statvfs. Ak operácia skončila chybou, tieto čísla môžu byť náhodné.

- **READDIR** Slúži na prečítanie obsahu priečinka. Argumentom je cesta k priečinku. Prvým riadkom odpovede je počet položiek nachádzajúcich sa v danom priečinku. Nasleduje pre každú položku jeden riadok s jej menom.

- **OPEN** Slúži na otvorenie súboru pre čítanie, resp. zápis. Argumentami sú cesta k súboru a mód, s ktorým sa má otvoriť. Odpoveďou je buď kladné číslo označujúce deskriptor otvoreného súboru na strane Androidu, alebo záporný kód chyby.

- **READ** Slúži na čítanie zo súboru. Argumentami sú deskriptor otvoreného súboru, maximálny počet prečítaných bytov a offset, od ktorého sa má z daného

súboru čítať. Prvým riadkom odpovede je počet prečítaných bytov, druhým riadkom odpovede je base64 zakódovaný reťazec s danými bytmi.

- **WRITE** Slúži na zápis do súboru. Argumentami sú deskriptor otvoreného súboru, počet bytov, ktoré treba zapísať, offset, od ktorého sa má do daného súboru zapisovať a reťazec v base64 kódovaní obsahujúci dáta, ktoré treba zapísať. Odpoveďou je kladný počet zapísaných bytov alebo záporný kód chyby.

- **CLOSE** Slúži na zatvorenie otvoreného súboru. Argumentom je deskriptor otvoreného súboru. Odpoveďou je odpoveď z systémového volania close() zavolaného na strane Androidu.

- **CREATE** Slúži na vytvorenie súboru. Argumentom je cesta k súboru a mód, s ktorým sa má vytvoriť. Odpoveďou je deskriptor otvoreného súboru.

- **MKDIR** Slúži na vytvorenie priečinka. Argumentom je cesta k priečinku a mód, s ktorým sa má vytvoriť. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **TRUNCATE** Slúži na zmenu veľkosti daného súboru (zväčšenie alebo zmenšenie). Argumentom je cesta k súboru a jeho nová veľkosť. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **UNLINK** Slúži na zmazanie daného súboru. Argumentom je cesta k súboru. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **RMDIR** Slúži na zmazanie daného priečinka. Argumentom je cesta k priečinku. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **RENAME** Slúži na premenovanie, resp. presunutie daného súboru, resp. priečinka. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **ACCESS** Slúži na otestovanie, či je súbor otvoriteľný s daným módom. Argumenty sú rovnaké ako pri operácii OPEN. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby.

- **UTIMENS** Slúži na zmenu času posledného prístupu a modifikácie súboru. Argumenty sú zložené z dvoch dvojíc, prvá reprezentuje čas posledného prístupu, druhá čas poslednej modifikácie. Jedna dvojica argumentov je zložená z dvoch čísel, prvé číslo vyjadruje čas ako počet sekúnd od začiatku epochy (1. 1. 1970 0:00 UTC) a druhá počet nanosekúnd od celej sekundy. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby. Túto operáciu nepodporujú všetky

zariadenia s OS Android (závisí od súborového systému na Androide a verzie Androidu).

- **FALLOCATE** Slúži na rezervovanie miesta v súborovom systéme pre daný súbor. Argumenty sú deskriptor otvoreného súboru, mód prístupu, offset od ktorého má byť miesto vyhradené a počet bytov na vyhradenie. Odpoveďou je 0 ak sa operácia vykonala úspešne alebo kód chyby. Túto operáciu nepodporujú všetky zariadenia s OS Android (závisí od súborového systému na Androide a verzie Androidu).

5.7 Rozšírenie pre prenos dát zo senzorov

Android poskytuje API pre prístup k dátam zo senzorov implementované podľa návrhového vzoru Observer. Keďže čítanie dát zo senzorov môže byť veľmi náročné na spotrebu batérie, aplikácia sa v momente, keď chce pristupovať k senzorom, musí zaregistrovať v systémovom SensorManageri. Keď už prístup k senzorom nie je potrebný, aplikácia sa môže odregistrovať. Protokol sme preto navrhovali tak, aby bol kompatibilný s takýmto spôsobom používania senzorových API na Androide.

Rozšírenie pre prácu so senzormi v štandardnej implementácii, priloženej k práci, je hlavne ukážkou riešenia tohto problému, a podporuje iba prácu so senzorom rotácie (gyroskopu) telefónu. Dostalo v protokole identifikátor ROTATIONSENSOR. Jeho súčasťou sú nasledovné operácie:

- **STARTSENSORS** Operácia nemá žiadne argumenty ani odpoveď. Slúži na začatie monitorovania všetkých senzorov.

- **GETROTATION** Operácia nemá žiadne argumenty. Odpoveďou sú 3 riadky, na prvom je poloha v X-ovej súradnici, na druhom poloha v Y-ovej súradnici a na treťom poloha v Z-ovej súradnici. Všetky polohy sú udané ako desatinné číslo od -1 po 1. Krátku dobu (rádovo v desiatkach ms) po zavolaní STARTSENSORS sú údaje zo senzora ešte nedostupné (senzor sa inicializuje). Počas tejto doby táto operácia vracia 3 nuly. Pred prvým zavolaním STARTSENSORS operácia vracia 3 nuly. Medzi zavolaním QUITSENSORS a prípadnom opätovnom zavolaní STARTSENSORS operácia vracia posledné namerané hodnoty.

- **QUITSENSORS** Operácia nemá žiadne argumenty ani odpoveď. Slúži na zastavenie monitorovania všetkých senzorov.

Kapitola 6

Implementačné detaily

6.1 Implementované súčasti

V rámci tejto práce boli pre OS Linux implementované knižnica libalc pre komunikáciu protokolom ALC, aplikácia pre prenos šifrovacieho kľúča pre protokol ALC pomocou vykreslenia QR kódu, aplikácia pre vyhľadanie aktívnych serverov protokolu ALC na lokálnej sieti a ukážková aplikácia ku knižnici libalc. Pre OS Android bola implementovaná aplikácia pre komunikáciu protokolom ALC podporujúca prenos šifrovacieho kľúča nasnímaním QR kódu a spravovanie uložených šifrovacích kľúčov.

6.2 Implementácia FUSE súborového systému

Podstatnou súčasťou tejto bakalárskej práce je aj implementácia samotného FUSE súborového systému. V tejto sekcii sa budeme venovať implementačným detailom jednotlivých funkcií, ktoré treba implementovať na to, aby náš súborový systém vyzeral plne funkčne.

6.2.1 getattr a statfs

getattr

int getattr(const char *, struct stat *);

Táto funkcia je podobná unixovej operácii stat(). Jej úlohou je naplniť štruktúru typu stat informáciami o súbore danom cestou v adresárovom podstrome. Štruktúra typu stat je definovaná informáciami, ktoré má obsahovať (ID zariadenia, na ktorom sa súbor nachádza, číslo inode v súborovom systéme, pod ktorým sa súbor nachádza,

prístupové práva k danému súboru, počet hard linkov odkazujúcich na súbor, ID používateľa a skupiny, ktorí vlastnia daný súbor, ID zariadenia, ak súbor predstavuje zariadenie, celkovú veľkosť v bytoch, veľkosť bloku súborového systému na ktorom sa súbor nachádza a počet blokov, ktoré súbor v súborovom systéme zaberá, časy posledného prístupu, zmeny a zmeny stavu) a ich poradím.

Všetky z týchto vlastností súboru sú štandardizovaným spôsobom vyjadriteľné číslom. Avšak na rôznych platformách je v tejto štruktúre na rôzne vlastnosti vyjadrený rôzny počet bytov. Preto sme sa rozhodli, že táto štruktúra bude v našom protokole prenášaná textovo a nie binárne.

statfs

int statfs(const char *, struct statvfs *);

Táto funkcia sa správa úplne rovnako ako getattr() akurát nenapĺňa štruktúru typu stat ale štruktúru typu statvfs. Implementovaná bola podobne.

6.2.2 readdir

int readdir(const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);

Táto funkcia je volaná, keď FUSE filesystém obdrží volanie readdir(). Ako parametre obdrží cestu k priečinku, ktorý systém chce prečítať, buffer, do ktorého sa ukladá zoznam súborov, plniacu funkciu, ktorá do buffra ukladá mená súborov po jednom, offset od ktorého chceme zoznam súborov začať čítať a informácie o otvorenom priečinku. Môže byť implementovaná dvomi spôsobmi:

- Plniaca funkcia je volaná s offsetom súboru v zozname súborov v priečinku a zoznam súborov sa môže plniť na viacero volaní readdir
- Offset súboru nie je pri volaní plniacej funkcie vyplnený (dáva sa doňho NULL) a zoznam súborov je prečítaný na jedno volanie readdir

Pre jednoduchosť implementácie a snahu o minimalizovanie počtu sieťových operácií bol vybratý druhý variant implementácie.

6.2.3 Práca s otvorenými súbormi (open, read, close, write, create, fallocate)

Tieto funkcie môžu byť vo FUSE súborovom systéme implementované dvomi spôsobmi:

- Volanie open() otvorí súbor a vráti file descriptor, číslo reprezentujúce daný súbor v tabuľke otvorených súborov. Volanie close() daný súbor zavrie a vymaže z tabuľky otvorených súborov. Volania read, write a fallocate dostávajú file descriptor ako položku v argumente typu fuse_file_info.
- Volanie open() iba kontroluje, či sa daný súbor dá otvoriť a volanie close() nič nerobí. Každé jedno zavolanie funkcie read, write, alebo fallocate na strane servera volá open() a close()

Zo snahy o urýchlenie prenosu súborov sme sa rozhodli pre prvý spôsob implementácie.

open

int open(const char *path, struct fuse_file_info *fi);

Táto funkcia dostane ako argumenty cestu k súboru, a v štruktúre typu fuse_file_info uložený mód, s akým sa súbor otvára (povolené možnosti sú rovnaké ako pri volaní funkcie open() zo štandardnej knižnice jazyka C, s tým, že ak je implementovaná FUSE funkcia create() tak by FUSE funkcia open() nemala byť nikdy zavolaná s módom vytvorenia súboru.

release

int release(const char *path, struct fuse_file_info *fi);

Funkcia na zavretie otvoreného súboru sa vo FUSE volá release. Táto funkcia dostane ako argument file descriptor a jej cieľom je ho zavrieť. Návratová hodnota tejto funkcie je ignorovaná. FUSE po zavolaní tejto funkcie už file descriptor nepoužije.

read a write

int read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi);
int write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi);

Tieto funkcie majú tri argumenty: file descriptor ukazujúci na súbor, z ktorého treba čítať, offset, od ktorého chce používateľ čítať dáta a počet dát, ktoré chce používateľ prečítať. Najväčší súvislý blok je nastaviteľný, v 32-bitovom systéme je zhora ohraničený veľkosťou 32 stránok.

create

int create(const char *path, mode_t mode, struct fuse_file_info *fi);

Táto funkcia slúži na vytvorenie súboru a jeho otvorenie na zápis. Argumenty sú rovnaké ako pri volaní štandardnej funkcie creat() jazyka C. File descriptor je zapísaný do štruktúry fuse_file_info a funkciu sme implementovali presne tak isto ako je popísané v manuále k funkcii creat[4].

fallocate

int fallocate(const char *path, int mode, off_t offset, off_t len, struct fuse_file_info *fi);

Táto funkcia je ekvivalentná Linuxovej funkcii fallocate(), preto bola implementovaná volaním tejto natívnej funkcie v Androide.

6.2.4 Ostatné funkcie (rename, access, rmdir, unlink, truncate, mkdir)

int unlink(const char *path);
int rmdir(const char *path);
int access(const char *path, int flags);
int mkdir(const char *path, mode_t mode);
int truncate(const char *path, off_t offset);
int rename(const char *oldpath, const char *newpath);

Tieto funkcie sú implementovateľné úplne priamočiaro, pre všetky je odpoveďou buď 0 alebo chybový kód podľa hlavičkového súboru errno.h so záporným znamienkom a všetky boli implementované tak, že sa po sieti iba poslali argumenty, vykonala sa prislúchajúca Linuxová funkcia s rovnakým názvom a ako výsledok bola vrátená buď 0 alebo hodnota errno.

6.3 Implementácia zdieľanej knižnice a podporných aplikácií pre Linux

6.3.1 Práca s UUID

UUID celý čas ukladáme vo forme textových reťazcov, na generovanie sme naprogramovali jednoducho použiteľné funkcie kompatibilné so štandardom RFC 4122[5].

6.3.2 Generovanie QR kódu

Obsah QR kódu, ktorý je následne snímaný Android aplikáciou tvoria dva textové reťazce oddelené medzerou. Na zobrazenie QR kódu do terminálu používame externú open source aplikáciu qrencode. qrencode síce má C++ API na vytváranie QR kódov, kód na zobrazovanie QR kódov do terminálu nie je však súčasťou tohto API iba rovnomennej CLI aplikácie[15].

6.3.3 Konfiguračné súbory

Konfiguračné súbory ukladáme do vlastnej zložky alc v /.config/ (tu sídlia konfiguračné súbory množstva iných Linuxových aplikácií). Formát konfiguračných súborov bol zvolený v súlade s Linuxovou filozofiou aby bol čo najjednoduchší a čitateľný pre bežného používateľa. Používame 2 konfiguračné súbory:

- **uuid.conf** V tomto súbore sa nachádza UUID (v textovej forme) daného používateľa sediaceho za daným počítačom.
- **keys.conf** V tomto súbore sa pre každé spárované zariadenie nachádza riadok s jeho UUID a šifrovacím kľúčom, oddelenými medzerou.

6.3.4 Šifrovanie

Na šifrovanie komunikácie používame knižnicu EVP zo sady knižníc OpenSSL. Na generovanie kryptograficky bezpečných náhodných inicializačných vektorov pre algoritmus AES-CBC používame knižnicu RAND zo sady knižníc OpenSSL.

6.4 Implementácia Android aplikácie

6.4.1 Štruktúra aplikácie

Aplikácia bola implementovaná tak, aby mala čo najjednoduchšiu štruktúru, skladá sa z 2 aktivít a 1 služby:

- **Hlavné menu** Obsahuje tlačítka, ktorými sa používateľ vie dostať k ostatným aktivitám, spustiť proces párovania, prípadne zapnúť server na komunikáciu so software na Linuxe.

- **Zoznam spárovaných zariadení** Spustiteľná z hlavného menu. Zobrazuje zoznam spárovaných zariadení, ku každému zobrazuje UUID a poskytuje možnosť jednoducho zariadenia rozpárovať. Šifrovacie kľúče z bezpečnostných dôvodov zobrazené nie sú.

- **Server** Služba spustiteľná z hlavného menu. Pri svojom spustení spustí dve vlákna: v jednom beží server, ktorý akceptuje prichádzajúce spojenia a spúšťa ich v samostatnom vlákne, v druhom aplikácia oznamuje UDP broadcastmi svoje UUID.

6.4.2 Práca s QR kódom

Na nasnímanie QR kódu sme sa rozhodli využiť externú open source aplikáciu Barcode Scanner (súčasť projektu ZXing). Autori tejto aplikácie poskytujú jednoduché API, ktoré sa postará o poslanie Intent-u, navrátenie výsledku a prípadne zobrazenie výzvy na inštaláciu aplikácie Barcode Scanner používateľovi, ak táto aplikácia nie je nainštalovaná. Spracovanie výsledku riešime metódou v hlavnom menu.

6.4.3 Šifrovanie spojenia a Base64 kódovanie

Na šifrovanie spojenia a generovanie kryptograficky bezpečných náhodných inicializačných vektorov používame štandardnú Java knižnicu javax.crypto. Na kódovanie dát do formátu Base64 sme si naprogramovali vlastný Wrapper, keďže štandardná Android funkcia pre kódovanie do Base64 je konfigurovateľná a v štandardnej konfigurácii vkladá znak nového riadka za každými 76 znakmi.

Kapitola 7

Dokumentácia

7.1 Kompilácia a inštalácia Linuxových nástrojov

Na skompilovanie linuxových nástrojov potrebujete mať nainštalované tieto nástroje

- CMake
- Kompilátor podporujúci C++11 a C99 designated initializers (napríklad Clang)
- OpenSSL knižnice
- FUSE knižnice

Na použitie linuxových nástrojov je potrebné mať nainštalované OpenSSL a FUSE. Na použitie aplikácie `alcpair` je navyše potrebné mať nainštalovanú aplikáciu qrencode [15]

Na skompilovanie treba najskôr mať pripravený kompilátor a zadať postupne príkazy `cmake .` a `make`. Nainštalovať všetky aplikácie a knižnice pre všetkých používateľov sa dá použitím príkazu `sudo make install`

Ukážkový príklad kompilácie a inštalácie:

```
export CC=/usr/bin/clang
export CXX=/usr/bin/clang++
cmake .
make -j8
sudo make install
```

7.2 Inštalácia Android aplikácie

Android aplikácia je dodávaná ako súbor vo formáte APK. Takéto súbory sa dajú na OS Android priamo nainštalovať, ak je na zariadení povolená inštalácia aplikácií z externých zdrojov.

7.3 alcpair

`alcpair` je aplikácia určená na spárovanie dvoch zariadení. Požaduje ako argument IP adresu na ktorej je telefón s nainštalovanou Android aplikáciou. Voliteľne môžete špecifikovať aj port, na ktorom na zariadení beží server. Príklad použitia: `alcpair -i 192.168.47.100 -p 6000`.

7.4 findphone

`findphone` je aplikácia určená na vyhľadanie zariadení so spusteným serverom na sieti. Nepožaduje žiadne argumenty. Príklad výstupu:

```
Waiting for at least 1 phone.... Press Ctrl+C to interrupt
Found devices:
192.168.47.100 d632615e-84b9-448c-92ac-fd8ce55bd7b8 PAIRED
```

7.5 androidfuse

`androidfuse` je aplikácia určená na pripojenie súborového podstromu z OS Android do prípojného bodu v súborovom systéme OS Linux pomocou FUSE. Požaduje ako argument IP adresu a prípojný bod. Voliteľne môžete špecifikovať aj port, na ktorom na zariadení beží server a priečinok z OS Android, ktorého obsah sa má pripojiť. Ak priečinok nie je špecifikovaný, prípája sa priečinok `/sdcard/`, na ktorom sa štandardne nachádza predvolené externé úložisko. Príklad použitia: `androidfuse -i 192.168.47.100 -p 6000 -r /sdcard1 -t /mnt/android/`

Kapitola 8

Návrhy na zlepšenie

Počas práce sa objavilo mnoho nápadov, ktorými by sa protokol ALC a jeho referenčná implementácia dali zlepšiť. Jedna z hlavných vecí, v ktorých protokol zaostáva je rýchlosť. Počas testovania sa nám nepodarilo dosiahnuť väčšie rýchlosti ako 1MB/s, čo je dosť málo, aj na to, že používame WiFi. Referenčná implementácia obsahuje funkcie na prístup k jedinému senzoru. Počas posielania dát (čítaní a zapisovaní) sa dáta dvakrát kódujú do Base64, čo podstatne zväčšuje objem posielaných dát a je pravdepodobne jednou z príčin pomalosti aplikácie. Aplikácia na párovanie vyžaduje k behu externú aplikáciu qrencode. Keďže qrencode je open source, dala by sa používaná časť zdrojového kódu so súhlasom autora skopírovať a použiť v rámci našej aplikácie. Pozorného čitateľa určite napadnú aj ďalšie nápady.

Ďalšia vec, čo aplikácii chýba je dôkladné odladenie a testovanie či je aplikácia naozaj bezpečná. Vo veľa častiach kódu totižto aplikácia nepredpokladá, že by dostala neplatný príkaz a ak náhodou takýto dostane, aplikácia spadne.

Záver

V tejto práci sa nám podarilo navrhnúť a implementovať funkčný prototyp protokolu pre prenos súborov a senzorických dát medzi OS Android a OS Linux a k nemu integráciu do Linuxového súborového systému pomocou FUSE, nástroj na vyhľadávanie Android zariadení na lokálnej sieti a systém pre prenos šifrovacieho kľúča pomocou QR kódu.

Popri práci sa vyskytli rôzne problémy, niektoré z nich sa podarilo vyriešiť, iné by si ešte zaslúžili dodatočnú analýzu alebo vylepšenie riešenia.

Táto práca slúži ako vhodný základ pre komplexnejší protokol, ktorý by zvládol spravovať zariadenie s OS Android prostredníctvom jednoduchého programovateľného rozhrania zo strany zariadenia s OS Linux.

Literatúra

[1] Android external storage permissions explained. [Citované 2015-05-22] Dostupné z http://possiblemobile.com/2014/03/android-external-storage/.

[2] How to use the new sd card access api in android lollipop. [Citované 2015-05-22] Dostupné z http://stackoverflow.com/questions/26744842/how-to-use-the-new-sd-card-access-api-presented-for-lollipop.

[3] How total commander writes to sd card in android 4.4. [Citované 2015-05-22] Dostupné z http://www.ghisler.ch/board/viewtopic.php?t=34834&start=15.

[4] Linux manual. [Citované 2015-05-28] Dostupné z http://linux.die.net/man/3/creat.

[5] Rfc 4122. [Citované 2015-05-28] Dostupné z https://www.ietf.org/rfc/rfc4122.txt.

[6] Google. Android api guides: Fundamentals. [Citované 2015-05-22] Dostupné z http://developer.android.com/guide/components/fundamentals.html.

[7] Google. Android api guides: Permissions. [Citované 2015-05-22] Dostupné z http://developer.android.com/guide/topics/security/permissions.html.

[8] Google. Android api guides: Process lifecycle. [Citované 2015-05-22] Dostupné z http://developer.android.com/guide/topics/processes/process-lifecycle.html.

[9] Google. Android api guides: Sensors overview. [Citované 2015-01-18] Dostupné z http://developer.android.com/guide/topics/sensors/sensors_overview.html.

[10] Google. Android debug bridge. [Citované 2015-03-04] Dostupné z http://developer.android.com/tools/help/adb.html.

[11] Google. Android ndk. [Citované 2015-05-22] Dostupné z https://developer.android.com/tools/sdk/ndk/index.html.

[12] Google. Statistics about relative number of devices running a given version of the android platform., January 2015. [Citované 2015-01-28] Dostupné z `http://developer.android.com/about/dashboards/index.html#Platform`.

[13] USB Implementers Forum Inc. Universal serial bus mass storage class specification overview. [Citované 2015-03-04] Dostupné z `http://www.usb.org/developers/docs/devclass_docs/Mass_Storage_Specification_Overview_v1.4_2-19-2010.pdf`.

[14] USB Implementers Forum Inc. Media transfer protocol v.1.1 specification, 2011. [Citované 2014-12-04] Dostupné z `http://www.usb.org/developers/docs/devclass_docs/MTPv1_1.zip`.

[15] Fukuchi Kentaro. Qrencode. [Citované 2015-05-28] Dostupné z `http://fukuchi.org/works/qrencode/`.

[16] OpenSource. Fuse documentation. [Citované 2015-05-22] Dostupné z `http://fuse.sourceforge.net/doxygen/index.html`.

[17] Oracle. Java se documentation: Jni types and data structures. [Citované 2015-05-22] Dostupné z `http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/types.html`.

[18] Photographic and Imaging Manufacturers Association Inc. Photography - electronic still picture imaging - picture transfer protocol (ptp) for digital still photography devices. [Citované 2015-03-04] Dostupné z `http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2012/jmv87/site/files/pima15740-2000.pdf`.

[19] Wikipedia. Comparison of file systems. [Citované 2015-01-28] Dostupné z `http://en.wikipedia.org/wiki/Comparison_of_file_systems#Supporting_operating_systems`.