SMS brána a systém núdzového vzdialeného prístupu prostredníctvom siete GSM pre OS Linux

Bakalárska práca

Danica Zajacová

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Informatika 9.2.1

Vedci: RNDr. Jaroslav Janáček

Bratislava, 2010

Abstrakt

Cieľom mojej bakalárskej práce je navrhnúť a implementovať SMS bránu spolu s núdzovým systémom vzdialeného prístupu prostredníctvom mobilnej siete GSM/UMTS pod OS Linux. Cieľom je vytvorenie sady programov, ktoré umožnia aplikáciam odosielanie SMS správ prostredníctvom pripojeného mobilného telefónu a zároveň umožnia vzdialený prístup (vrátane call back a GPRS/EDGE/UMTS dátového prístupu) pre administrátora v prípade nedostupnosti primárnych sieťových kanálov.

Kľúčové slová: SMS brána, GSM/UMTS, vzdialený prístup pre administrátora

Čestne prehlasujem, že som túto bakalársku prácu vypracovala samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Touto cestou by som sa chcela poďakovať môjmu školiteľovi RNDr. Jaroslavovi Janáčkovi, za peknú a zaujímavú tému, zapožičanie technických pomôcok a pomoc pri tvorbe tejto práce.

Obsah

Úvod

Jednu z mnohých nepríjemností, ktorá sa môže stať správcovi servera, predstavuje aj situácia, keď jemu do opatery zverený server prestane komunikovať so svetom. Ak je server dôležitý (čo taký server zvyčajne býva), správcovi neostáva nič iné, len čo najrýchlejšie prísť k serveru, diagnostikovať problém a pokúsiť sa server znovu spojazdniť. Výraz prísť k serveru v sebe zahŕňa prejsť 10, 100 metrov, pricestovať z Popradu do Bratislavy alebo sa urýchlene vrátiť z dovolenky na Kanárskych ostrovoch. Najmä v posledných dvoch(pre niekoho aj troch) prípadoch je situácia maximálne nežiadúca a okrem času nás slovo prísť stojí aj nemalé peniaze. Ideálne by bolo, ak by správca nemusel nikam cestovať a mohol by problém vyriešiť na diaľku. Najlepšie tak, že by sa mu nejako podarilo pripojiť k nekomunikujúcemu serveru. Spojenie klasickým spôsobom(zvyčajne cez ethernetovú linku) zo servera do internetu nefunguje, a preto treba hladať iné náhradné riešenie ako sa k serveru pripojiť.

V dnešnej dobe takéto náhradné riešenie pre veľký server môže poskytnúť aj malý, vcelku bežný a dokonca ani nie úplne najmodernejší a najdrahší mobil. Okrem toho, že z mobilu dokážeme telefonovať a posielať sms správy, dokážeme sa dnes bez problémov z mobilu pripojiť na internet a využívať všetky jeho dostupné prostriedky. Ba čo viac, my môžeme pripojiť náš mobil k počítaču a pripojiť sa z počítaču cez mobil na internet. Mobil nám v tomto prípade poslúži ako starý dial-up modem. Rovnako, ako sa dokážeme pripojiť na internet, dokážeme z počítaču cez mobil telefenovať a posielať sms správy.

Takže ak pripojíme mobil k serveru(alebo server k mobilu) a server stratí spojenie zo svetom cez klasický a rýchly ethernetový kábel, existuje tu malá šanca, že sme úplne nestratili spojenie so serverom a administrátor tak vďaka mobilu budem mať šancu sa k vypadnutému serveru znovu pripojiť. Celá predstava ako by to mohlo celé fungovať, je nasledovná:

- majme server a k nemu pripojený mobil
- server nech je klasicky pripojený na internet
- vznikol problém a server stratil spojenie s internetom

- server pošle cez mobil administrátorovi sms, že nastal problém - v sms ho môže blyžšie špecifikovať
- administrátor si prečíta sms a prezvoní mobil pripojený k serveru
- server zistí, že ho prezváňal administrátor a spustí aplikáciu, ktorá zabezpečí, že sa server pripojí cez mobil na internet a pomocou OpenVPN sa ho podarí pripojiť k inému serveru. Tento server mu poskytne adresu, pomocou ktorej bude adresovateľný z pripojeného serveru. Pripojený server takto umožní prístup k inak nedostupnému serveru a administrátor sa bude môcť na inak nedostupný server pripojiť a začať diagnostiku.
- administrátor sa pripojí na server a začne diagnostiku

Prvá časť tejto práce sa zaoberá vytvorením prepojenia medzi mobilom a počítačom. Po vytvorení prepojenia sa práca venuje riešeniu problému, ako poslať sms z počítača na mobil a ako zistiť zo strany počítača, že na mobil pripojený k počítaču volá administrátor. Zistené poznatky sú zúžitkované pri návrhu a tvorbe programu MDemon. Tento program beží na pozadí systému a plní dve hlavné úlohy. Na jednej strane slúži ako sms brána, pomocou ktorej môžu užívatelia a programy posielať sms správy, na druhej pracuje ako program moniturujúci a obsluhujúci prichádzajúce hovory. Tieto hovory ukončuje a na základe čísla volajúceho spúšťa v novom procese inú aplikáciu alebo nerobí nič. Činnosť programu sa dá ovládať pomocou iných aplikácií, ktoré posielajú programu MDemon príkazy. Pomocou týchto príkazov sa MDemon dá korektne ukončiť, odpojiť a spätne pripojiť k mobilu, alebo využiť k odoslaniu sms správy na zvolené číslo. Základné princípy jeho fungovania popisuje tretia kapitola tejto práce. Štvrtá kapitola sa zaoberá pripojením počítača na internet pomocou mobilu a riešením problematiky vzdialeného prístupu k počítaču, ktorý nemá verejnú IP adresu. Posledná časť práce je venovaná programom umožňujúcim ovládanie programu MDemon a dvom aplikáciam, využívajúcim program MDemon na odoslanie SMS správy v prípade výpadku pripojenia na internet a vytvoreniu možnosti vzdialeného prístupu administrátorovi využitím programov WvDial a OpenVPN.

Kapitola 1

Prepojenie mobilu s PC

Táto kapitola je venovaná pripojeniu počítača k mobilu. Na ten sa chceme pripojiť, aby sme mohli pomocou neho odosielať sms správy, spravovať prichádzajúce hovory a pripojiť sa na internet. Tieto funkcie v mobile zabezpečuje GSM/GPRS modem a pripojením sa naň máme väčšinou možnosť tieto funcie využívať. Existuje tu zaužívaná tradícia z čias minulých, že k modemom sa dá pristupovať pripojením sa na sériový port modemu pomocou sériového káblu(RS-232). Pri vzniku nových technológii však vznikla akási potreba nahradiť sériové káble niečim iným - napríklad USB alebo Bluetooth. Aby sa nemuseli prepisovať programy, ktoré vedeli pracovať iba so sériovým portom a k modemom sa ľahko pridala možnosť byť pripojený pomocou inej technológie, vytvorila sa v nových technológiach možnosť emulácie sériového prepojenia. Takto si napríklad aplikácie na počítači a modem myslia, že spolu komunikujú starým spôsobom cez sériový kábel, ale pritom môžu byť reálne prepojené napríklad pomocou USB alebo Bluetooth.

Dnes sa na prepojenie PC a mobilu využíva najmä Bluetooth a USB. Preto sa v nasledujúcich častiach tejto kapitoly bedeme venovať prepojeniu pomocou Bluetooth a USB. V oboch prípadoch sa budeme snažiť pripojiť počítač na sériový port mobilu, teda samotný modem a budeme emulovať prepojenie pomocou sériového kábla.

1.1 Bluetooth

Bluetooth je bezdrôtová komunikačná technológia slúžiaca na prenos informácií na krátke vzdialenosti. Veľkosť vzdialenosti na akú môžu bluetooth zariadenia komunikovať záleží okrem prostredia a okolitých javov aj od samotných zariadení. Zariadenia podľa výkonu rozdeľujeme do troch tried. Do prvej triedy patria zariadenia s výkonom až 100 mW a teoreticky sú schopné

prenášať informácie na vzdialenosť 100 metrov. Do druhej triedy patria zariadenia využívajúce až 2,5 mW majúc dosah až 10 metrov a do tretej triedy zariadenia s výkonom 1 mW pokrývajúce vzdialenosť do 1 metra. Väčšina bluetooth zariadení ako mobilné telefóny, laptopy a headsety spadá do druhej triedy.

Objem dát, ktorý sa v súčastnosti dá preniesť pomocou Bluetooth sa pohybuje niekde v rozmedzí do 3 Mbit/s v závislosti od typu jednotlivých zariadení. Rýchlosť prenosu dát samozrejme závisí od pomalšieho zariadenia. Pekné na technológií Bluetooth je, že si zachováva spätnú kompatibilitu. Novšie zariadenia sú vďaka tomuto schopné komunikovať s tými staršími.

Bluetooth zariadenia operujú v okolí 2,4-GHz frekvenčného pásma, ktoré sa ďalej rozdeľuje na 79 kanálov 1 MHz širokých. Na rozdiel od Wifi, bluetooth zariadenie si na komunikáciu nevyberá len jeden kanál, ale náhodne strieda kanály 1600 krát za sekundu. Navzájom komunikujúce zariadneia samozrejme menia pásmo rovnako, aby mohli vysielať a prímať informácie na rovnakých frekvenciách.

Dve a viac navzájom komunikujúcich zariadení tvorí *piconet*. Jeden piconet môže pozostávať až z 8 zariadení. Aby zariadenia mohli navzájom komunikovať, musia používať v rovnakom čase rovnaké frekvencie. O toto sa stará jedno zariadnie tvoriace tzv. *master of piconet*. Zvyšné zariadenia sú označované ako *slaves*(otroci). Master of piconet má 2 úlohy. Nariaďuje ostatným zariadeniam, aké frekvencie majú používať a stará sa o to, kedy môže ktoré zariadenie komunikovať. Bluetooth zariadenia komunikujú pomocou vzájomného striedania sa. Každú chvílu môže komunikovať niekto iný. Jedno zariadenie teoreticky môže byť súčasťou viac ako jedného piconetu. Takýchto 2 a viac piconetov tvorí *scatternet*. V praxi však veľa zariadení nepodporuje možnosť bytia súčasťou viacerých piconetov.

1.1.1 Identifikácia zariadenia

Každé bluetooth zariadenie, ktoré bolo kedy vyrobené obsahuje unikátnu 48 bitovú bluetooth adresu nazývanú *Bluetooth adresa*. Táto adresa je obdobou MAC adries pre ethernet, ale na rozdiel od MAC adries sa používa vo všetkých vrstvách komunikácie medzi zariadeniami. Rovnako ako prideľovanie MAC adries, aj pridelovanie Bluetooth adries spadá pod IEEE Registration Authority. Okrem svojej adresy môže mať každé bluetooth zariadenie aj svoje meno, s ktorým navonok vystupuje (napr.: „Moj mobilik“). Toto meno však nemusí byť unikátne a slúži len pre jednoduchšiu identifikáciu zariadenia človekom.

1.1.2 Viditeľnosť a pripojitelnosť jednotlivých zariadení

Bluetooth zariadeniam sa dá ďalej nastaviť aj akým spôsobom majú reagovať na okolitý svet. Slúžia im na to dva parametre, ktoré nadobúdajú hodnoty on alebo off: Inquiry scan a Page scan. *Inquiry scan* vyjadruje schopnosť byť videný ostatnými zariadeniami. *Page scan* určuje ochotu prijímať príchodzie spojenia. Ich použitie a vzájomný vzťah najlepšie vyjadruje nasledujúca tabuľka.

Inquiry scan	Page scan	Interpretácia
On	On	Zariadenie je detekovateľné a ochotné prijímať žiadosti o spojenie.
Off	On	Cudzie zariadenia, ktoré nepoznajú adresu môjho zariadenia nie sú schopné moje zariadenie detekovať. Zariedenie je však ochotné prijímať žiadosti o spojenie zariadeniam, ktoré poznajú jeho bluetooth adresu.
On	Off	Zariadenie je detekovateľné, ale odmieta prijímať žiadosti o spojenia.
Off	Off	Zariadenie je nedetekovateľné a odmieta s hocikým komunikovať.

1.1.3 Autentifikácia zariadení

Ak chcú spolu zariadenia komunikovať prostredníctvom bluetooth, musia sa najprv nájsť a navzájom autentifikovať. Zariadenia sa autentifikujú tak, že si navzájom potvrdia akési zdieľané tajomstvo(shared secret). Toto zdieľané tajomstvo tvorí *PIN* pozostávajúci zo sekvencie až 12 alfanumerických znakov pomocou, ktorého sa ďalej šifruje komunikácia medzi zariadeniami. PIN slúži na generovanie tzv. *link key*. Link key sa uloží na všetkých zariadeniach a pomocou neho sa pošlú zašifrované dáta, ktoré druhá strana zase dešifruje. Ak bluetooth zariadenie A pošle zariadeniu B zašifrovaný paket a strana B ho dešifruje, je jasné, že autentifikácia prebehla úspešne. Je dobré si uvedomiť, že počas procesu autentifikácie PIN nikdy nie je prenesený vzduchom. Ochrana pomocou PINu patrí medzi ľahký až stredný typ ochrany, takže ak niekto veľmi chce čítať komunikáciu medzi zariadeniami, tak pravdepodobne bude úspešný. Ľudia na zakódovanie komunikácie väčšinou používajú jednoduchý PIN typu „1234“ alebo „0000“, čo tvorí bezpečnostný problém. Druhý problém je, že do zariadení ako headset sa nedá vyťukať PIN, a preto vyzužívajú často ako prednastavený(default) PIN „0000“. Tieto problémy sa snaží

vyriešiť technológia Bluetooth 2.1., pomocou tzv. *Simple Pairing* procedúr. Užívateľ tu už nie je zaťažovaný vymýšľaním a zadávaním PIN, ale je mu iba položená otázka, či sa chce spárovať s konkrétnym zariadením. PIN je automoticky vygenerovaný zariadením. Simple Pairing so sebou navyše prináša aj bezpečnejšie metódy ohľadom šifrovania akými sú SSH, IPSec, PGP a SSL.

1.1.4 Bluetooth profily a protokoly

Keď už vieme adresu bluetooth zariadenia na ktoré sa chceme pripojiť a máme na zariadeniach nastavené, že sú ochotné komunikovať, je dobré sa začať zaoberať otázkou, kam sa to presne ideme pripojiť a ako budú zariadenia medzi sebou komunikovať. Toto riešia bluetooth transportné protokoly a bluetooth profily(služby) implementované nad jednotlivými protokolmi. Pri klasickom sieťovom(TCP/IP) programovaní máme definované sieťové služby(napr. HTTP), ktoré na prenos dát využívajú transportné protokoly (napr. v prípade HTTP TCP a UDP). Ak sa zariadenie A chce pripojiť na zariadenie B a využívať nejakú jeho konkrétnu službu(HTTP), tak zariadenie B si vytvorí napr. streamovy[1] serverovský socket. Socket pripojí na nejaký port(pri HTTP je to port 80) a na tom porte bude počúvať prichádzajúce spojenia a poskytovať službu. Zariadenie A si vytvorí klientský streamový socket, zistí adresu zariadenia B a port na ktorom počúva. Následne sa pokúsi pripojiť svoj klientský socket na adresu B a port, na ktorom služba B počúva a poskytuje službu, po ktorej túži A. Služba na B akceptuje(alebo neakceptuje) spojenie a zariadenia si môžu(alebo nemôžu) pomocou dohodnutých pravidiel(protokolu) vymieňať informácie. Rovnako to funguje aj pre bluetooth. Zariadenie B(v našom prípade mobil) poskytuje profily(služby), pričom pre každú službu čo poskytuje, má vyhradený kanál(port). Tieto profily na komunikáciu využívajú transportné protokoly(RFCOMM, L2CAP,...) podobne ako sieťové služby využívajú TCP a UDP. Zariadenie A(náš počítač) sa snaží pripojiť na bluetooth profil(v našom prípade Dial-up networking) zariadenia B naviazaný na nejaký kanál a na komunikáciu budú používať protokol(napr RFCOMM).

RFCOMM(Radio Frequency Communications) je spoľahlivý streamový protokol ponúkajúci rovnaké služby a garancie ako TCP protokol. Jeho cieľom bolo predovšetkým emulovať RS-232 sériový port, aby výrobcovia mohli ľahko pridať bluetooth vlastnosti do svojich existujúcich zariadení pracujúcich so sériovým portom(napr. taký modem mobilu). Na rozdiel od TCP, kde

[1] to znamená, že pri výmene dát sa bude používať TCP protokol. Ak by si vytvoril datagramový socket, tak na výmenu dát sa bude používať UDP protokol

serverovské aplikácie majú k dispozícii 65 535 portov, serverovské aplikácie využívajúce RFCOMM ich majú k dispozícií len 30.[2] Ďalej RFCOMM nemá žiadne porty vyhradené pre konkrétnu službu(profil) ako napríklad TCP má vyhradený port 80 pre webové služby. Klientské aplikácie chcúce využívať nejakú služby si takto musia zistiť, na ktorý port sa majú pripojiť. Toto našťastie rieši SDP protokol, ktorý je popísaný nižšie.

Ďalšími protokolmi slúžiacimi na prenos dát sú protokoly L2CAP a ACL. L2CAP(Logical Link Control and Adaptation Protocol) je paketový protokol, ktorý môže byť nakonfigurovaný v rôznych stupňoch spoľahlivosti, tak že v sebe enkapsuluje protokol RFCOMM alebo sa správa ako UDP. L2CAP má k dispozíci až 16 383 portov, z toho 2048 vyhradených. Jeho hlavnou nevýhodou je, že nastavením doručovacej politiky na jednom z L2CAP spojení, sa automaticky nastavia aj všetky zvyšné L2CAP spojenia so zariadením, na ktoré sme pripojení. Navyše prenastavíme si tak aj RFCOMM spojenia, pretože každé RFCOMM spojenie je zapúzdrené v L2CAP spojení. ACL(Asynchronus Connection-oriented Logical) protokol zas slúži na zapúzdrenie L2CAP pripojení. Dve navzájom prepojené zariadenia môžu mať medzi sebou len jedno ACL prepojenie. ACL sa málokedy používa na čistý prenos dát a využíva sa skôr na zaobalenie paketov vyšších protokolov. Na prenos dát sa najviac používa RFCOMM protokol, pričom mnoho bluetooth rozhraní na prenos dát využíva len RFCOMM. A keďže vo svojej bakalárskej práci chcem spoľahlivo streamovať dáta, rozhodla som sa vo svojej práci používať práve tento protokol.

Z Bluetooth zariadení sa dá zistiť, aké profily poskytujú a na základe podpory týchto profilov vieme určiť, ako sme schopní dané zariadenie využiť. Ďalej vieme zistiť aké transportné protokoly môžme využívať pri chcení využiť konkrétny profil. Existencia niektorých profilov býva často závislá na existencií iných - napr. DUN je závislí na SPP. Bluetooth profily tvoria akýsi súbor pravidiel, podľa ktorých sa musí dané zariadenie správať. Pre ilustráciu uvádzam niektoré často využívané bluetooth profily.

- *Object Push Profile*(OPP): Umožňuje zariadeniam posielať dáta(objekty) akými sú napríklad súbory, obrázky či hudba.
- *File Transfer profile*(FTP): Umožňuje jednému zariadeniu prezerať súborový systém iného zariadenia a následne si na ňom mazať, sťahovať, premenovávať, kopírovať a nahrávať súbory.

[2]Oficiálne sa pri bluetooth tieto porty nenazývajú portami, ale kanálmi(channels), takže po správnosti by sa malo povedať, že RFCOMM má k dispozícii 30 kanálov. Pretože sú to len inak pomenované porty, budem ich ďalej nazývať portami a občas kanálmi.

- *Serial Port Profile*(SPP): Profil umožňujúci emulovať sériové spojenie medzi zariadeniami. Využívaný ako náhrada za sériový kábel.
- *Dial-up Networking Profile*(DUN): Poskytuje dosiahnutie internetu a iných dial-up služieb pomocou Bluetooth. Najdôležitejší profil pre túto bakalársku prácu a vlastne jediný, ktorý v svojej práci budem využívať. Jeho fungovanie je závislé na SPP.

SDP(Service Discovery Protocol) je protokol umožňujúci klientskym aplikáciam sa pripojiť na správny port, ak chcú využívať nejakú konkrétnu službu napríklad Dial-up networking. Celé to funguje nasledovne: Každé bluetooth zariadenie má nejaký známy(„well-known“) port, na ktorom počúva SDP server. Serverovská aplikácia, ktorá chce využívať nejaký port sa musí so svojim popisom zaregistrovať na SDP serveri, ktorý k danej aplikácii priradí port na ktorom bude odteraz počúvať. Potom príde klientska aplikácia a opýta sa SDP servera, kde má hľadať konkrétnu službu. SDP server ju odkáže na príslušný port alebo zreferuje, že takú službu neposkytuje. Každá služba by mala mať jedinečné 128 bitové UUID(Universaly Unique Identifier), ktoré ju jednoznačne identifikuje. Klientská aplikácia sa takto pýta na konkrétne UUID. Služba Dial-up networking má napríklad UUID 0x1103. Služby sa v bluetooth terminológii označujú ako profily, takže služba dial-up networking je vlastne dial-up networking profile.

1.1.5 Bluetooth pod OS Linux

Aby sme mohli pohodlne využívať bluetooth technológie pod nejakým systémom, je dobré aby preň existovalo akési bluetooth rozhranie inak nazývané aj *Bluetooth Stack*. Takéto rozhranie pre operačné systémy Linux tvorí *BlueZ* a jeho jadro sa stalo dokonca súčasťou linuxového kernelu od verzie 2.4.6. Pre zaujímavosť, Ubuntu 9.10, ktoré pri písaní tejto práce používam, používa kernel verzie 2.6.31.

BlueZ sa zaoberá ovládaním jednotlivých bluetooth zariadení(adaptérov), ktoré Linux využíva. Na jednoduchú manipuláciu s týmito zariadeniami ponúka knižnice(C a Pythone) pre programátorov a sadu programov pre užívateľov, ktoré sa dajú volať z konzoly. Pre účely tejto práce nám postačia práve tieto programy a využijeme ich na nastavenie lokálneho bluetooth adaptéra, ktorý mienime používať a na vytvorenie spojenia medzi počítačom a mobilom.

1.1.6 Pripojenie počítača na mobil v Linuxe

Prvé čo musíme urobiť je nastaviť si na počítači lokálny bluetooth adaptér, ktorý budeme využívať na pripojenie sa k mobilu. Na to nám slúži príkaz hciconfig, ktorý má takúto štruktúru:

```
# hciconfig <zariadenie> <command> <argumenty.....>
```

Pri zadaní hciconfig bez argumentov sa nám zjavia informácie o všetkých bluetooth adaptéroch s ich aktuálnymi nastaveniami, ktoré systém využíva.

```
# hciconfig
hci0: Type: USB
BD Address: 00:1F:E2:E4:97:F8 ACL MTU: 1021:8 SCO MTU: 64:1
UP RUNNING PSCAN
RX bytes:1025 acl:0 sco:0 events:36 errors:0
TX bytes:1597 acl:0 sco:0 commands:36 errors:0
```

V tomto konkrétnom prípade náš systém využíva len jeden bluetooth adaptér hci0 s bluetooth adresou 00:1F:E2:E4:97:F8. Adaptér je prístupný(UP RUNNING) pričom ma zapnutý Page Scan(PSCAN) a vypnuty Inquiry Scan (ISCAN). Zvyšné údaje nie sú pre nás momentálne dôležité.

Náš hci0 adaptér môžeme vypnúť(zneprístupniť) pomocou príkazu

```
# hciconfig hci0 down
```

a opätovne zapnúť pomocou

```
# hciconfig hci0 up
```

Pomocou príkazu hciconfig a jeho parametrov sme schopní ďalej zobraziť a nastaviť meno nášho adaptéru a nastaviť mu hodnoty pre Inquiry Scan a Page scan. Pre ďalšie nastavenia bluetooth adaptéru odporúčam manuálové stránky, ktoré sa vyvolajú nasledovne:

```
# man hciconfig
```

Ak nevieme bluetooth adresu mobilu, na ktorý sa chceme pripojiť, pomôže nám to zistiť nástroj hcitool. Hcitool s príkazom scan, nám zobrazí detekovateľné bluetooth zariadenia v dosahu nášho adaptéru. Ak hcitool nenájde náš mobil, pravdepodobne na ňom treba nastaviť, aby bol detekovateľný pre ostatné zariadenia a skúsiť príkaz znovu. V nasledujúcom príklade hcitool s príkazom scan našiel náš mobil a notebook nejakého Pala.

```
# hcitool scan
Scanning ...
11:22:33:44:55:66 Moj mobilik
AA:BB:CC:DD:EE:FF Palov notebook
```

V ďalšom kroku chceme náš počítač pripojiť na Dial-up network profile(DUN) nášho mobilu. Tento profil nám umožní sa pripojiť na GSM/GPRS modem nášho mobilu a prístup do siete internet. Na to aby sme sa mohli pripojiť na DUN, musíme zistiť na ktorom kanáli(porte) mobil ponúka danú službu a či vôbec. Na to slúži príkaz sdptool. Pomocou neho si môžme nechať vypísať všetky profily, ktoré mobil podporuje spolu s kanálmi na ktorých ich poskytuje alebo sa len opýtať, či a kde poskytuje danú službu. Pri druhom prístupe treba zadať UUID profilu.

```
# sdptool browse 00:0E:ED:57:A7:38
Browsing 00:0E:ED:57:A7:38 ...
Service Name: Dial-up networking
Service RecHandle: 0x10002
Service Class ID List:
  "Dialup Networking" (0x1103)
  "Generic Networking" (0x1201)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Dialup Networking" (0x1103)
    Version: 0x0100

Service Name: COM 1
Service RecHandle: 0x10004
Service Class ID List:
  "Serial Port" (0x1101)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
```

```
  base_offset: 0x100

# sdptool search 0x1103
Class 0x1103
Inquiring ...
Searching for 0x1103 on 00:0E:ED:57:A7:38 ...
Service Name: Dial-up networking
Service RecHandle: 0x10002
Service Class ID List:
  "Dialup Networking" (0x1103)
  "Generic Networking" (0x1201)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Dialup Networking" (0x1103)
    Version: 0x0100
```

Teraz vieme, že mobil ponúka službu na kanáli 1, takže sa ideme pripojiť na kanál 1. Pomocou príkazu rfcomm sme schopní sa pripojiť na kanál jeden a to tak, že linux bude vnímať mobil ako zariadenie, ktoré je k nemu pripojené pomocou sériového portu. Po zadaní príkazu

```
# rfcomm bind 0 00:1F:01:29:5E:38 1
```

nám v adresári /dev vznikne uzol rfcomm0[3], ktorý bude predstavovať náš mobil a my k nemu budeme môcť pristupovať ako k zariadeniam pripojeným pomocou sériového portu. Príkaz bind nám nevytvorí spojenie medzi počítačom a mobilom, ale vytvorí iba akési viazanie. Spojenie vznikne až, keď sa nejaký program pokúsi otvoriť /dev/rfcomm0 zariadenie. Ak by sme namiesto bind dali connect, počítač vytvorí s mobilom spojenie, ktoré sa bude dať prerušiť stlačením CTRL-C. Pre účely tejto práce je connect nežiadúci, pretože ak sa pokúsim pomocou daemona otvoriť /dev/rfcomm0 program vráti chybu: „Device or resource busy“ a preto používam bind. Viazanie medzi počítačom a mobilom cez /dev/rfcomm0 sa da zrušiť pomocu príkazu:

```
# rfcomm release /dev/rfcomm0
```

[3]ak by sme namiesto 0 dali napríkld 5, vznikne uzol rfcomm5

Posledná vec, ktorú sme ohľadom spojenia medzi mobilom a počítačom nespomenuli, je autentifikácia. Tá prebehne až sa počítač pokúsi vytvoriť spojenie(connect) s mobilom. Pre naše účely je veľmi žiadúce, aby mobil umožňoval počítaču sa na neho automaticky pripojiť(toto by sa v mobili dalo dať lahko nastaviť) a zároveň aby sa automaticky vedeli spárovať, pretože inak každý pokus o connect treba ručne povoliť, čo nechceme.

1.2 USB

USB(Universal serial bus) je štandard sériovej zbernice určenej najmä na pripojenie periférií k počítaču. Jeho zámerom bolo aj je nahradiť mnoho druhov sériových a paralelných portov. Dnes možno pomocou USB pripojiť množstvo zariadení ako myši, klávesnice, tlačiarne, skenery, flash pamäte, externé hardisky, či mobilné telefóny. Okrem toho sa využíva aj na nabíjanie batérií menších zariadní akými sú MP3 prehrávače a tiež mobilné telefóny.

USB momentálne podporuje 4 rýchlosti na prenos údajov:

- **Low-speed** - predstavuje rýchlosť 1.5 Mbit/s a je využívaná HID(Human Interface Devices) zariadenimi ako myši, klávesnice a joystiky.
- **Full-speed** - predstavuje rýchlosť až 12 Mbit/s, ktorá prišla s USB 1.1
- **High-speed** - predstavuje rýchlosť až 480 Mbit/s, ktorá prišla v roku 2001 s USB 2.0
- **SuperSpeed** - predstavuje rýchlosť až 4.8 Gbit/s, ktorá prišla nedávno s USB 3.0

Verzie jednotlivých USB by mali byť spätne kompatibilné, takže port USB 2.0 by mal byť schopný komunikovať so zariadeniami 1.1 a USB 3.0 s USB 2.0 a 1.1. Rýchlosť komunikácie medzi zariadeniami samozrejme určuje to pomalšie.

„USB systém má asymetrický design pozostávajúci z hostiteľského kontroléra(angl. host controller) a viacerých zariadení spojených v uzavretom cykle. Do cyklu môžu byť zapojené prídavné rozbočovače (angl. hub), pričom môžu tvoriť až 5-úrovňové stromy na jeden kontrolér.“ [3] Na jeden kontolér môže byť pripojených maximálne 127 zariadení vrátane rozbočovačov.

Jedno USB zariadenie môže pozostávať z viacerých logických podzariadení(subdevices), ktoré vyjadrujú funkcie USB zariadenia. Jedno zariadenie môže ponúkať viacero funkcií, ako napríklad mobilný telefón môže v sebe obsahovať 2 funkcie: prístup k modemu a prístup k pamäti telefónu. Takéto zariadenie sa nazýva zložené zariadenie a každá jeho funkcia predstavuje jedno logické

podzariadenie, ktorému je priradená USB adresa identifikujúca zariadenie. Tieto logické podzariadenia sú pripojené k rozbočovaču zariadenia. Rozbočovač zariadenia je potom pomocou USB kábla a konektora pripojený do USB portu ďalšieho zariadenia(napr. počítač). Hostiteľský kontrolér priradí každej funkcií zariadenia(logickému podzariadeniu) jednu USB adresu a pristupuje sa k nej ako jednému zariadeniu. Pripojením mobilu z príkladu na náš počítač sme defacto pripojili 2 zariadenia: modem a pamäť telefónu.

Na identifikáciu funkcionality zariadenia USB štandard definuje kódy tried, čo umožňuje zariadeniam načítať driver založený na danej funkcionalite zariadenia a tvorcovi driveru podporu zariadení od rôznych výrobcov, ktorý dodržujú štandardy. Príkladmi takýchto tried sú:

- **Human Interface Device (HID)** - určená pre periférne zariadenia akými sú napríklad myš a klávesnica
- **Printer** - určená pre tlačiarne
- **Mass Storage** - určená pre zariadenia ako USB flash drive, čítačky pamäťových kariet, digitálnu kameru, externé hardisky
- **Communications and CDC Control** - určená pre modemy a ethernet adaptéry

1.2.1 USB a Linux

V dnešných Linuxoch môžu byť USB zariadenia používané hneď po pripojení. Po pripojení mobilu pomocou USB sa pripojilo k nášmu PC s bežiacim Linuxom niekoľko linuxových zariadení. Nás zaujíma zariadenie mobilného telefónu predstavujúce adaptér sériového portu. To bude predstavovať zariadenie /dev/ttyUSB* (zvyčajne /dev/ttyUSB0), ktoré vzniklo s pripojením mobilu k počítaču.

Na získanie prehľadu o USB zberniciach a zariadeniach na ne pripojených slúži nástroj *lsusb*, ktorý sa dá vyvolať priamo ako príkaz v konzole. Pri výskyte problému s USB zariadeniami alebo zariadením je dobré si pozrieť posledné „debug“ informácie generované linuxovým jadrom hneď po pripojení zariadenia. To sa dá zavolaním príkazu *dmesg*.

Kapitola 2

Komunikácia medzi počítačom a modemom mobilu

Mobilné telefóny komunikujú s okolitým svetom pomocou bezdrôtových GSM, GPRS a iných im podobných sietí spravovaných jednotlivými operátormi. Prácu s týmito sieťami umožňuje mobilným telefónom špeciálny modem, ktorý funguje len s pripojenou SIM kartou vydanou mobilným operátorom. SIM karta(subscriber identity module card) v sebe obsahuje IMSI (service-subscriber key), ktorého cieľom je identifikovať používateľa mobilného zariadenia. Tento špeciálny modem býva práve GSM modem, ktorý v sebe často podporuje aj GPRS, EDGE ci 3G/HSDPA technológie.

GSM modem sa vo svojej podstate správa podobne ako obyčajný dial-up modem s tým rozdielom, že neposiela dáta cez fixnú telefónnu linku, ale odosiela a prijíma dáta pomocou rádiových vĺn. Z počítača sa dá ovládať pomocou AT príkazov, pričom GSM modem podporuje rovnakú sadu štandardných AT príkazov ako dial-up modem. Z toho vyplýva, že sa dá použiť rovnako ako bežný dial-up modem. Okrem štandardnej sady AT príkazov, GSM modemy podporujú aj rozšírenú sadu týchto príkazov, ktorá umožňuje využívanie ďalších služieb akými sú napríklad odosielanie a príjem sms, či monitorovanie sily signálu. Tieto rozšírené AT príkazy sú definované v GSM štandardoch.

V predchádzajúcej kapitole sme riešili otázku ako sa pripojiť na GSM modem mobilu pomocou Bluetooth alebo USB. To sa nám podarilo vyriešiť tak, že niekde v systéme máme znakové zariadenie reprezentujúce sériový port mobilu, pomocou ktorého sme schopní komunikovať s modemom mobilu. Po otvorení tohto zariadenia na čítanie a zápis sme schopní komunikovať s GSM modemom ako s hociktorým iným znakovým zariadením. S GSM modemom sa dá komunikovať pomocou vyššie spomínaných AT príkazov.
O AT príkazoch a ich využití na nastavenie modemu, odosielanie SMS správ

a spravovanie telefonického hovoru budú pojednávať nasledujúce časti tejto kapitoly.

2.1 História AT príkazov

Do začiatku 70-tych rokov minulého storočia modemy typicky operovali nad priamo-vytáčacími(direct-dial) telefónnymi linkami nasledujúcim spôsobom: Užívateľ zvyčajne pred pripojením sa manuálne vytočil telefónne číslo, kam sa chcel pripojiť, alebo zdvihol telefón v prípade, že zvonil. V niektorých prípadoch počítač sám vytočil číslo pomocou vytáčača(dialer) pripojeného k počítaču (zvyčajne pomocou RS-232 portu).

V 70-tych rokoch minulého storočia, keď prišla revolúcia mikropočítačov, počítačový priemysel hľadal spôsob ako povedať modemu, aby vytočil číslo pomocou softvéru. Toto sa vedci snažili docieliť rôznymi spôsobmi, z ktorých najrozumnejší priniesla v roku 1977 firma Hayes Communications so svojím produktom Smartmodem. Smartmodem komunikoval s počítačom pomocou sady AT príkazov vymyslených firmou Hayes Communications, pričom jeho hlavnou výhodou od iných riešení bolo aj to, že sa pripájal k počítaču pomocou vtedy velmi rozšírenou RS-232. Ich modem pracuje v dvoch módoch:

- **Dátový mód**(Data mode) - len preposiela odoslané a prijímanné dáta
- **Príkazový mód**(Command mode) - dáta sú prijímané ako príkazy pre lokálny modem, ktoré modem následne spracúva.

Na prepnutie z dátového módu na príkazový mód sa využívala sekvencia znakov „+++“, po ktorých nasledovala aspoň sekundová pauza. Z príkazového módu sa prepínalo do dátového zase pomocou príkazu „0“. Mnoho príkazov toto prepnutie do dátového módu robilo automaticky, takže nebolo treba vždy zadať príkaz „0“.

Sada AT príkazov od Hayes Communications(tzv. Hayes command set) obsahovala množstvo príkazov na manipuláciu telefónnej linky(vytáčanie, skladanie slúchadla...) a príkazy na nastavenie modemu. Hayes Communications nevidal na Hayes command set žiadne vlastnícke práva, ale patentoval si len čas, ktorý modem čaká po „+++“. To zapríčinilo rozmach AT príkazov aj na modemy iných spoločností a Hayes command set sa stal štandardom. Zvyšovaním prenosových rýchlostí a požiadaviek na modemy, začali vznikať ďalšie sady AT príkazov tzv. rozširujúce sady príkazov a rôzne ďalšie štandardy. Jemný poriadok do sveta chaosu neskôr priniesol štandard TIA/EIA-602, ktorý bol vybudovaný na Hayes extended set používajúcih prevažne príkazy s „&“.

2.2 Syntax písania AT príkazov rozšírených o GSM/UMTS príkazy

- AT príkazy môžu byť písané malými alebo veľkými písmenami.
- Každý príkazový reťazec musí začínať s „AT“ alebo „at“ okrem príkazov „A/“ a „+++“. „At“ alebo „aT“ sú brané ako chyba.
- Každý príkazový reťazec musí obsahovať menej ako 40 znakov.
- Každý príkazový reťazec musí byť zakončený <CR> okrem „+++“ a „A/“
- Jeden príkazový reťazec môže v sebe obsahovať viacero príkazov.
- Telefónne číslo môže pozostávať len z nasledujúcich znakov: 1 2 3 4 5 6 7 8 9 * = , ; # + > . Ostatné znaky(napr. medzera a podčiarkovník) budú odignorované.
- Príkazy obsahujúce číselný parameter môžu byť použité bez číselnej hodnoty. V tom prípade budú prezentované s nulovou hodnotou.
- Ak príkazový reťazec obsahuje dva po sebe idúce príkazy bez parametrov, modem odpovie s errorovou hláškou
- po zadaní príkazu „ATZ“, musí byť rešpektovaná pauza aspoň 2 sekundy.
- Ak sa vyskytne chyba pri písaní príkazu, mala by byť odstrániteľná pomocou backspase klávesy.
- GSM/UMTS príkazy používajú syntaktické pravidlá pre rozšírujúce príkazy
- Každý rozširujúci príkaz má aj svoju testovaciu variantu, kedy sa za príkaz uvedie „=?“, čím sa testuje podporovanosť daného príkazu modemom. Ak modem daný príkaz podporuje, modem vráti zoznam podporovaných subparametrov pre daný príkaz. Ak nie, vypíše „ERROR“.
- Rozširujúce príkazy s parametrom majú aj tzv čítaciu variantu s „?“, ktorou sa zisťuje aktuálna hodnota parametra. Spúšťacie príkazy, čítaciu variantu nemajú.
- Rozširujúce príkazy sa oddelujú bodkočiarkou.

- Ak sú zapnuté verbálne odpovede pomocou „V1“, tak v prípade úspešného vykonania všetkých príkazov príkazového reťazca modem vráti „<CR><LF>OK<CR><LF>“. V prípade neúspechu pre nejaký príkaz z reťazca vráti „<CR><LF>ERROR<CR><LF>“ a nevykoná zvyšné príkazy.

- Ak sú zapnuté numerické odpovede pomocou „V0“, tak v prípade úspešného vykonania všetkých príkazov príkazového reťazca modem vráti „0<CR>“. V prípade neúspechu pre nejaký príkaz z reťazca vráti „+CME ERROR: <err>“ a nevykoná zvyšné príkazy. <err> predstavuje číslo chyby, ktorá nastala.

Nasledujúce 2 príklady budú obsahovať príklad príkazového reťazca a odpoveď modemu na daný príkazový reťazec vo verbálnej forme s vypnutým opakovaním príkazu[5].

```
ATCMD1 CMD2=12; +CMD1; +CMD2=,,15; +CMD2?; +CMD2=?<CR>
```

Príkazový reťazec(riadok) sa začína „AT“, nasleduje príkaz CMD1, príkaz zo sady rozširujúcich príkazov GSM/UTMS CMD2 s parametrom 12 ukončený a oddelený od +CMD1 bodkočiarkou obsahujúci predponu +C typickú pre GSM/UMTS rozširujúce príkazy, rozširujúci GSM/UMTS príkaz +CMD1 oddelený bodkočiarkou, GSM/UMTS príkaz +CMD2 vynechávajúci prvé 2 parametre a meniaci tretí parameter na 15, čítací príkaz pre +CMD2, ktorým chceme zistiť nastavené hodnoty parametrov a testovací príkaz pre +CMD2, pomocou ktorého sa chceme dozvedieť možné hodnoty parametrov, použitelných pri +CMD2 príkaze. Príkazový reťazec je ukončený znakom „carriage return“

```
<CR><LF>+CMD2: 3,0,15,"GSM"<CR><LF>
<CR><LF>+CMD2: (0-3),(0,1),(0-12,15),("GSM","IRA")<CR><LF>
<CR><LF>OK<CR><LF>
```

Prvý riadok odpovede modemu na predošlý obsahuje informácie o aktuálnom nastavení parametrov pre +CMD2, druhý hovorí o možnostiach nastavenia +CMD2 a tretí nás informuje o tom, že modem úspešne spracoval náš príkazový reťazec.

2.3 AT príkazy využité v tejto práci na nastavenie mobilu

AT

AT je skratka vyjadrujúca „attention“ a ako samotný príkaz slúži na overenie, či modem počúva.

ATZ

ATZ slúži na resetovanie modemu, pomocou ktorého všetky nastavenia nadobudnú pôvodné hodnoty.

E0

E zabezpečí aby nám modem nerobil echo - neposielal späť odpoveď s príkazom, ktorý sme mu zadali.

E1

E1 naopak od príkazu E0 nastaví, aby nám modem robil echo.

V1

V1 nastaví, aby nám modem posielal odpovede vo verbálnej podobe ako „OK“ a „ERROR“ a nie v číselnej, čo sa dá nastaviť pomocou príkazu „V0“

+CLIP

Ak je +clip vypnutý(teda jeho prvý parameter je nastavený na 0 alebo modem ho nepodporuje), prichádzajúci hovor nám modem mobilu oznámi len pomocou reťazca „RING“[1], ktorý zopakuje každým zazvonením. Ak je nastavený na 1, pri prichádzajúcom hovore sa každým zazvonením okrem reťazca „RING“ objaví aj „+CLIP:«tel.číslo>"“ s možnými prídavnými parametrami, podľa implementácie jednotlivým výrobcom. <tel.číslo> predstavuje telefónne číslo volajúceho. Ak volajúci si nepraje, aby sme mohli zistiť jeho telefónne číslo, modem vráti namiesto telefónneho čísla prázdny reťazec. Celé fungovanie najlepšie vysvetluje nasledujúci príklad:

[1] V prípade nastaveného +CRC na 1 (+CRC=1) sa pri každom zazvonení telefónu namiesto „RING“ objavý „+CRING: <typ>“, kde <typ> bude nahradený typom prichádzajúceho hovoru(VOICE, DATA, VOICE/DATA)

```
at+clip=0
OK

RING

RING
at+clip=1
OK

RING

+CLIP: "421908123456",145

RING

+CLIP: "421908123456",145

RING

+CLIP: " ",128
```

V príklade najprv nastavíme +clip na 0, resp clip necháme vypnutý. Následne nám niekto zavolá a po druhom zvonení zloží. Nie sme schopní zistiť, kto volal. Pred ďalším hovorom si nastavíme clip na 1. Teraz už vidíme, kto nám volá. Číslo ďalšieho volajúceho aj napriek zapnutému clipu nevidíme, lebo pred nami tají číslo.

+CMGF

Príkaz slúži na nastavenie módu v akom chceme manipulovať s SMS. V prípade nastavenia +CMGF na 1, budeme s SMS pracovať v textovom móde. V prípade nastavenia na 0 v PDU móde. Rozdiel medzi textovým a PDU módom bude vysvetlený pri odosielaní SMS.

2.4 Odoslanie SMS v textovom móde

Odosielanie SMS v textovom móde je z užívateľského hľadiska jednoduché a prirodzené na rozdiel od neskôr uvedeného PDU módu. Proces napísania a odoslania SMS v textovom móde vyzerá nasledovne: Modemu sa pošle príkaz,

```
+CMGS=<tel.číslo><CR>
```

kde namiesto <tel.číslo> sa napíše telefónne číslo adresáta SMS. Následne modem pošle späť

```
<CR><LF><greater_than><space><CR>
```

a až potom môže užívateľ začať zadávať text SMS. Ak pri zadávaní textu použijeme enter, modem nám zase pošle,

```
<CR><LF><greater_than><space><CR>
```

po ktorom budeme môcť pokračovať v písaní SMS. SMS sa pošle na odoslanie po napísaní znaku <ctrl-Z>. Proces tvorby SMS môže byť hocikedy prerušený vyslaním znaku <ESC>. Po úspešnom odoslaní SMS modem vypíše[2],

```
+CMGS: <mr>
```

kde <mr> predstavuje GSM 03.40 TP-Message-Reference v integer formáte, a oznámi nám, že všetko prebehlo úspešne(pomocou OK alebo 0). V prípade neúspechu pošle,

```
+CMS ERROR: <err>
```

kde <err> predstavuje číslo chyby, zdôvodňujúce, prečo sa odoslanie nepodarilo.

V nasledujúcom príklade sa pokúsime odoslať SMS s textom „Ahoj bobor“ na číslo +421908123456, čo sa nám očividne aj podarí:

```
AT+CMGS="+421908123456"<CR>
> Ahoj bobor<ctrl-Z>
+CMGS: 56

OK
```

V ďaľšom príklade sa budeme snažiť prerušiť odoslanie SMS s textom „Ahoj bobor“ na číslo +421908123456. SMS sa neodošle a modem vráti pozitívnu správu o prerušení, ktorá sa od správy o úspešnom odoslaní bude líšiť o riadok s „+CMGS: <mr>“.

```
AT+CMGS="+421908123456"<CR>
>Ahoj bobor<ESC>

OK
```

[2]V prípade, že +CSMS je nastavené na 1 a sieť dané nastavenie podporuje, pozitívna odpoveď má takýto tvar „+CMGS: <mr>,<scts>“, kde <scts> predstavuje GSM 03.40 TP-Service-Centre-Time-Stamp v time-string formáte

2.5 Odoslanie SMS v PDU móde

V PDU móde sa pri odosielaní SMS používa ten istý príkaz ako v textovom móde s tým rozdielom, že SMS sa tu posiela so všetkými hlavičkami a je zakódovaná ako binárny string pozostávajúci z hexadecimálnych IA5 znakov, ktoré reprezentujú jednotlivé oktety(osmice bitov) správy. Proces odoslania SMS v PDU móde si ukážeme na príklade v ktorom odošleme SMS s textom „Ahoj bobor“ na číslo +421908123456.

```
AT+CMGS=22<CR>
>0001000C912491802143650000A41F45B0D12BFC56F39<ctrl-Z>
+CMGS: 13

OK
```

Ako prvé sa odošle volanie +CMGS s príslušným parametrom. Ako parameter tu na rozdiel od textového módu neslúži telefónne číslo adresáta, ale číslo vyjadrujúce počet oktetov v správe bez prvého oktetu(každý oktet je vyjadrený hexadecimálne) tvoriaceho "00". Po odoslaní +CMGS príkazu, modem pošle späť

```
<CR><LF><greater_than><space><CR>
```

a my môžeme zadať SMS správu so všetkými hlavičkami v PDU formáte. Správa sa zakončí znakom <ctrl-Z> rovnako ako pri textovom móde. Ak nenastala chyba, SMS správa sa odošle. Po úspešnom odoslaní SMS modem vypíše[3],

```
+CMGS: <mr>
```

kde <mr> predstavuje GSM 03.40 TP-Message-Reference v integer formáte, a oznámi nám, že všetko prebehlo úspešne(pomocou OK alebo 0). V prípade neúspechu pošle „+CMS ERROR: <err>“. <err> predstavuje číslo chyby, zdôvodňujúce, prečo sa odoslanie nepodarilo.

Význam jednotlivých oktetov správy je popísaný v nasledujúcej tabuľke:

[3] V prípade, že +CSMS je nastavené na 1 a sieť dané nastavenie podporuje, pozitívna odpoveď má takýto tvar „+CMGS: <mr>,<scts>“, kde <scts> predstavuje GSM 03.40 TP-Service-Centre-Time-Stamp v time-string formáte

Veľkosť v oktetoch	Hodnota	Popis
1	00	Indikuje, že nenahrádzame SMSC číslo. Bude použité číslo, ktoré je nastavené v mobile.[4]
1	01	Tento oktet kontroluje ďalšie PDU nastavenia správy
1	00	Predstavuje referenciu na túto správu. V prípade, že by sa vyskytla chyba pri doručovaní, toto číslo by sa stalo súčasťou chybovej hlášky, čím by sme vedeli určiť SMS, pri ktorej nastala chyba
1	0C	Určuje počet číslic, ktoré obsahuje telefónne číslo adresáta
1	91	Hodnota 91 vyjadruje, že číslo adresáta bude v medzinárodnom tvare
6	249180214365	Predstavuje hodnotu telefónneho čísla adresáta tak trochu v divnom tvare, kde číslice sú v dvojiciach pozamieňané. 24 predstavuje 42, 91 predstavuje 19, 80 predstavuje 08, a tak ďalej. Ak by telefónne číslo pozostávalo z nepárneho počtu číslic, za poslednú číslicu by sa dalo F a tak by sa premiešalo číslo.(napríklad číslo 42190812345 by vjadrovalo zoskupenie znakov 2491802143F5)
1	00	Identifikátor protokolu. Toto sa môže použiť pri indikácií nejakého vyššieho protokolu. „0“ indikuje, že nebol použitý žiaden vyšší protokol
1	00	Reprezentuje dáta kódujúcu schému. 00 indikuje, že správa je zakódovaná pomocou GSM7 abecedy(používa sa ako defaultná). Okrem GSM7 abecedy sa zvikne ešte používať kódovanie UCS-2, ktorú reprezentuje oktet s hodnotou 08
1	0A	Reprezentuje dĺžku samotnej správy („Ahoj bobor“) v počte oktetov, ktorými je popísaná
10	41F45B0D12BFC56F39	telo samotnej správy („Ahoj bobor“) zakódovanej v GSM7 kódovaní prevedenej na oktety. Prevod správy na takúto správu si ukážeme v nasledujúcej časti.

Zakódovanie tela správy

Najprv treba správu zakódovať do povoleného formátu. V našom konkrétnom prípade kódujeme správu „Ahoj bobor“ do GSM 7 kódovania. 1 znak v GSM 7 kódovaní je zakódovaný siedmimi bitmy. Niektoré znaky v GSM kódovaní patria medzi tzv. špeciálne znaky. Tie sa kóduju pomocou 2 znakov, kde prvý je tzv. ecsape znak a druhý predstavuje špecialny znak. Majme teraz našu správu v GSM7 kódovaní a ukážme si jednotlivé znaky v binárnej podobe.

1000001	A
1101000	h
1101111	o
1101010	j
0100000	_
1100010	b
1101111	o
1100010	b
1101111	o
1110010	r

PDU mód, však správu kóduje po oktetoch, takže našu správu v binárnej podobe treba rozdeliť na oktety. To prebehne nasledovne: Zoberieme si prvé písmeno správy a pridáme k nemu na začiatok posledný bit ďalšieho písmena. Takže zoberieme binárny string 1000001 reprezentujúci A a posledný bit z h, čo je 0 a vyjde z toho 01000001, čo v hexadecimálnom zápise predstavuje 41. Teraz si zoberieme zvyšných 6 bitov z h a na začiatok k nim pridáme posledné 2 bity z o. Takýmto spôsobom budeme kódovať až neprídeme nakoniec, kedy k zvyšným bitom posledného písmena pridáme na začiatok potrebný počet núl, aby nám vyšiel posledný oktet.

01000001	41
11110100	F4
01011011	5B
00001101	0D
00010010	12
10111111	BF
11000101	C5
01101111	6F
00111001	39

Kapitola 3

Základné princípy fungovania programu MDemon

V prvej kapitole sme riešili otázku, ako sa pomocou Bluetooth alebo USB pripojiť na sériový port mobilu, na ktorom počúva GSM modem. To sa nám podarilo vyriešiť tak, že teraz máme niekde v systéme znakové zariadenie reprezentujúce sériový port mobilu, pomocou ktorého sme schopní komunikovať priamo s modemom mobilu. V druhej kapitole sme si povedali ako komunikovať s GSM modemom mobilu. Ďalšiu časť tejto práce tvorí program MDemon, ktorý v sebe využíva poznatky z prvých 2 kapitol. Princípy jeho fungovania sú predmetom tejto kapitoly.

Úlohou programu MDemon je počúvať modem mobilu, či neprichádza hovor od nejakého konkrétneho čísla a ak áno tak vykonať nejakú určitú úlohu - najlepšie spustiť nejaký iný program. MDemon má byť navyše ovládateľný inými aplikáciami a má umožňovať, aby ho tieto aplikácie mohli ovládať aj cez internet. MDemon má byť samozrejme ovládateľný len oprávnenými aplikáciami. Aplikácie oprávnené ovládať MDemon by mali byť schopné odpojiť a pripojiť progmram MDemon k mobilu a zároveň by mali byť schopné pomocou tohto programu odoslať sms na nimi zvolené číslo. Odpojenie programu MDemon od mobilu tvorí nevyhnutný predpoklad pre iné aplikácie, ako napríklad PPP daemon, ktoré sa na pripájanie do siete internet prostredníctvom mobilu potrebujú sami seba pripojiť na modem mobilu. Pripojený program MDemon k mobilu tvorí zas nevyhnutný predpoklad pre počúvanie správ od mobilu a posielanie SMS správ pomocou tohto programu.

Pri náhľade na program MDemon netreba zabúdať na zámer, s ktorým bol vytvorený. Pomocou neho sa má administrátorovi servera oznámiť výpadok pripojenia servera na internet a zároveň má pomôcť administrátorovi serveru, na ktorom beží MDemon, pripojiť sa cez mobil k tomuto serveru. Predpokladaný scénar udalostí na ktoré je MDemon stavaný je nasledovný:

- Majme server so spusteným programom MDemon pripojeným na modem mobilu.
- Server klasicky pripojený na internet zrazu stratí pripojenie do siete internet.
- Server pošle prostredníctvom programu MDemon sms správu administrátorovi servera, že nastal problém, ktorý v sms správe môže blyžšie špecifikovať.
- Administrátor si prečíta sms správu a prezvoní mobil pripojený k serveru.
- MDemon zistí, že administrátor volá na mobil, ukončí hovor a spustí program, ktorý sa má spustiť v tejto situácií.
- Spustený program odpojí program MDemon od telefónu a zabezpečí administrátorovi vzdialene sa pripojiť na server prostredníctvom mobilu.
- Po skončení tohto pripojenia nejaký program serveru znovu pripojí program MDemon k modemu mobilu.

Po naštudovaní si scénara predpokladaných udalostí pre program MDemon je ľahko vidieť, že by bolo vhodné spravovať odpájania a pripájania programu MDemon k mobilu tak, aby nenastal napríklad aj takýto sled udalostí:

- MDemon zistí, že administrátor volá na mobil, ukončí hovor a spustí program, ktorý sa má spustiť v tejto situácií.
- Spustený program odpojí program MDemon od telefónu aby mohol spustiť PPP daemona na pripojenie servera k sieti internet prostredníctvom modemu mobilu.
- Medzi odpojením programu MDemon od mobilu a spustením PPP daemona, príde iná aplikácia a opätovne pripojí MDemon k mobilu.
- Programu PPP daemon sa nepodarí pripojiť na modem mobilu a následne ani server k sieti internet cez mobil.

MDemon rieši manažment svojho odpájania sa a pripájania k mobilu tak, že pokiaľ je k programu MDemon pripojená aplikácia, čo odpojila MDemon od mobilu, tak žiadna iná aplikácia okrem nej samotnej nesmie obnoviť toto pripojenie. Iné aplikácie môžu pripojenie obnoviť až keď sa stratí spojenie s danou aplikáciou alebo odpájacia aplikácia obnoví pripojenie. V druhom

prípade sa vlastne znovu pripojenie neuskutoční, pretože MDemon už bude pripojený.

V prvej časti tejto kapitoly sa budeme zaoberať otázkou ako pripojiť program MDemon k modemu mobilu, ako naň posielať AT príkazy a ako od modemu prijímať odpovede. Následne budeme riešiť ako program MDemon rieši problém, aby bol dosiahnuteľný a ovládateľný oprávnenými aplikáciami bežiacimi aj mimo miestneho počítača. To sa vyrieši tak, že MDemon bude fungovať ako server na ktorý sa budú pripájať klientské aplikácie, ktoré sa budú snažiť pomocou dohodnutých príkazov ovládať program MDemon. MDemon bude tieto príkazy spracúvať a späť posielať odpovede. Otázkou ako pripojiť klientskú aplikáciu k mobilu sa zaoberá druhá časť tejto kapitoly, pričom protokol vzájomnej komunikácie rieši tretia. Štvrtá časť objasňuje ako je MDemon schopný pripájať klientské aplikácie, obsluhovať klientske aplikácie a zároveň počúvať a spracúvať správy od modemu mobilu.

3.1 Komunikácia medzi programom MDemon a modemom mobilu

Podstná časť programu MDemon spočíva v komunikácií medzi modemom mobilu a programom MDemon. Na jednej strane potrebujeme modemu mobilu posielať príkazy pre jeho nastavenie a posielanie SMS správ, na druhej ho potrebujeme počúvať, či naň práve neprichádza hovor zo známeho alebo aj neznámeho čísla.

Aby komunikácia bola vôbec možná, musíme najprv program MDemon pripojiť k modemu mobilu. Na modem mobilu sa program môže napojiť otvorením sériového portu mobilu, na ktorom modem počúva. V systéme by sme mali mať niekde špeciálny súbor reprezentujúci sériový port mobilu. Tento špeciálny súbor je typu *znakové zariadenie* a v Linuxe sa ním dá pracovať ako s každým iným znakovým zariadením.

Komunikácia so znakovým zariadením sa uskutočňuje pomocou posielania a prijímania znakov. Posielanie znakov sa vykonáva pomocou systémového volania *write* a čítanie za pomoci systémového volania *read*[1]. Aby sme však mohli so znakovým zariadením(súborom) narábať, musíme si ho najprv otvoriť pomocou systémového volania *open*. Systémové volania open, read a write sú súčasťou základných knižníc jazyka c definovaných v hlavičkových súboroch <fcntl.h>, <sys/stat.h>, <sys/types.h> a <unistd.h>.

[1]Program vlastne vníma znakové zariadenie a teda aj celý modem mobilu ako obyčajný súbor, z ktorého sa dajú čítať a zapisovať znaky.

3.1.1 Volanie open

```
int open(const char *pathname, int flags)
```

Toto volanie nám otvorí súbor ležiaci na mieste určenom pathname s istými príznakmi flags. Súbor môže byť otvorený s viacerými príznakmi, pričom každý príznak je v systéme vyjadrený pomocou nejakého čísla. Pomocou logického súčtu (alebo), môžme skonbinovať viacero týchto príznakov. Najdôležitejší a zároveň aj povinný príznak vyjadruje, či má byť súbor otvorený len pre čitanie (O_RDONLY), len pre zápis(O_WRONLY) alebo pre čítanie a zápis zároveň(O_RDWR). Kedže chceme modemu posielať príkazy a zároveň od neho spätne dostávať odpovede a správy identifikujúce prichádzajúci hovor, potrebujeme si zariadenie otvoriť na čítanie aj zápis súčasne. Funkcia open po vykonaní vráti číslo odteraz reprezentujúce otvorený súbor (File descriptor - súborový popisovač/deskriptor) alebo -1 v prípade, že sa súbor nepodarilo otvoriť.

Príklad volania pre otvorenie sériového portu modemu ležiacom na */dev/rfcomm0* na čítanie aj zápis s príznakom O_NONBLOCK, potrebným na to, aby operácia open hneď vrátila výsledok a nečakala kým bude zariadenie pripravené komunikovať, vyzerá nasledovne.

```
open("/dev/rfcomm0", O_RDWR | O_NONBLOCK);
```

Funkcia open po vykonaní vráti číslo odteraz reprezentujúce otvorený súbor (File descriptor) alebo -1 v prípade chyby, kedy nastaví, aby errno signalizovalo chybu.

3.1.2 Volanie write

```
ssize_t write (int fd, const void *buf, size_t n);
```

Write pre svoje fungovanie potrebuje vedieť súborový deskriptor(fd), aby vedel kam má zapisovať, buffer(buf), vyjadrujúci čo má zapisovať, a počet bajtov(n), ktoré má zapísať zo začiatku buffra(buf). Funkcia po vykonaní vráti počet bajtov, ktoré sa jej podarilo zapísať alebo -1 v prípade zlyhania. Príklad poslania prikazu „AT\r“ zariadeniu so súborový deskriptorom 24:

```
write(24, "AT\r", 3);
```

3.1.3 Volanie read

```
ssize_t read (int fd, void *buf, size_t n);
```

Read načíta zo súboru určeného deskriptorom (fd) nanajvýš n bytov a zapíše ich do buffra(*buf). Ako výsledok vracia koľko bajtov sa jej podarilo prečítať alebo -1 ak nastala nejaká chyba.

3.1.4 Odoslanie AT príkazu a získanie odpovede

S otvoreným sériovým portom mobilu na čítanie aj zápis môžme teraz bez problémov posielať modemu mobilu AT príkazy na nastavenie modemu pre naše účely a odoslanie sms správy. Po vyslaní AT príkazu, modem prijíme AT príkaz, spracuje ho a pošle späť odpoveď. Odpovede nás zaujímajú, pretože okrem ďalších informácií nás informujú, či spracovanie príkazu prebehlo úspešne.

Spracovanie AT príkazu môže modemu trvať rôzne dlho a rovnako rôzne dlho môže trvať aj doručenie odpovede programu MDemon. Okrem toho, mobil nám nemusí poslať hneď celú odpoveď, ale môže nám ju poslať po častiach. Takže ak program zavolá write pre poslanie AT príkazu a hneď za ním zavolá read na získanie odpovede, read nemusí prečítať celú odpoveď od mobilu.

Riešeniu tohto problému môže čiastočne volanie sleep(int n) medzi volanimi write a read, ktoré na n sekúnd pozastaví program. Mobil takto dostane n sekúnd čas aby nám poslal všetky časti správy, ktoré si systém uloží do buffru a po obnovení programu ich poskytne funkcí read. Toto riešenie však nie je dokonalé a niekedy n sekúnd nám nemusí stačiť a niekedy je n sekúnd zbytočne dlhá doba na čakanie. Ak vieme aspoň čiastočne určiť ako má odpoveď vyzerať, resp. najlepšie ako má končiť, môžme použiť cyklus pomocou ktorého budeme dovtedy volať funkciu read, pokiaľ nedostaneme uspokojujúcu odpoveď. Toto riešenie je síce správne, ale rovnako nie je veľmi uspokojivé, lebo neustále volanie read veľmi zaťažuje procesor.

Na odoslanie AT príkazu a spätné získanie odpovede využíva program MDemon svoju vlastnú funkciu *send_ command_ and_ get_ response*.

funkcia send_command_and_get_response

```
int send_command_and_get_response(char *command, char **response);
```

Funkcia sa volá s dvoma argumentami. Prvý argument obsahuje celý príkaz, ktorý sa odošle mobilu. Druhý parameter predstavuje miesto kam sa má prijatá odpoveď od mobilu zapísať. Funkcia po úspešnom odoslaní príkazu

a zaznamenaní odpovede vráti 0. V prípade, že stratila spojenie s mobilom vráti -1. Ak sa jej nepodarilo poslať príkaz vráti 1 a v prípade vnútorného zlyhania 2.

Táto funkcia používa na získavanie odpovedí od mobilu jemne odlišný prístup od predošlých riešení využitím systémového volania *select*. Toto volanie slúži na sledovanie a obsluhovanie viacero zariadení zároveň. Select po zavolaní čaká, pokiaľ sa neozve aspoň jedno z jeho sledovaných zariadení, alebo neprejde zadefinovaný čas(môže to byť aj nekonečno). Ak sa ozve nejaké zariadenie, dá sa zistiť, ktoré to bolo a patrične sa dá obslúžiť. Toto sledovanie však nerobí v nekonečných cykloch, ale s využitím operačného systému, ktorému sa samotný proces ozve, či má niečo na poslanie, či sa uvolnil pre zápis alebo sa dostal do chybového stavu. Týmto sa zaťaženie procesoru minimalizuje a on sa môže medzitým naplno venovať iným činnostiam. Takto môžeme sledovať aj len jediné zariadenie - modem mobilu, či už nemá pre nás odpoveď. Samozrejme mobil môže poslať len čiastočnú odpoveď, a preto aj volanie funkcie select MDemon zaobaľuje do nekonečného cyklu, kým mu nepríde uspokojivá odpoveď alebo nenastane nejaká výnimočná situácia(napr. strata spojenia medzi mobilom a počítačom). Funkciu select si detajlnejšie popíšeme neskoršie, keď budeme naraz sledovať viacero zariadení.

3.2 Pripojenie aplikácie k programu MDemon

MDemon rieši otázku pripojenia aplikácií k nemu samému pomocou internetových soketov. Tieto internetové sokety umožňujú pripojiť sa k programu MDemon aplikáciam bežiacim na lokálnom počítači, ale aj aplikáciam bežiacim na inom počítači pripojenom cez internetovú sieť. O internetových soketoch a ich použití pre naše účely hovoria nasledujúce časti tejto podkapitoly.

3.2.1 Internetový soket

Soket(angl. socket - zásuvka) predstavuje koncový bod komunikácie. Internetový soket je soket používaný na komunikáciu dvoch aplikácií cez internet prostredníctvom internetových adries pozostávajúcich z IP adresy a čísla portu. Aplikácia, ktorá chce komunikovať s inou aplikáciou prostredníctvom internetového soketu si musí najprv vytvoriť svoj internetový soket. Prostredníctvom tohto soketu bude neskoršie schopná komunikovať s druhou stranou rovnako ako s pripojeným znakovým zariadením. Na to aby však mohla komunikovať aplikácia s inou aplikáciou prostredníctvom soketu, musí svoj soket prepojiť so soketom inej aplikácie. Tu sa rozlišujú dva druhy apli-

kácií: klient a server, pričom klientská aplikácia sa snaží pripojiť svoj soketu k soketu servera. Aby soket servera bol dosiahnuteľný klientskému soketu, zviaže server svoj soket s nejakou adresou. V prípade, že chce byť server dosiahnuteľný cez internet, server si vytvorí internetový soket a ten zviaže s internetovou adresou. Na túto adresu sa potom snaží klientská aplikácia pripojiť svoj internetový soket. Aby pripojenie prebehlo úsepešne, server musí akceptovať klientskú aplikáciu.

V našom prípade bude teda MDemon fungovať ako server. Ako server si vytvorí internetový soket a zviaže ho s nejakou internetovou adresou počítača, na ktorom beží. Na túto adresu sa potom budú snažiť pripájať klientske aplikácie.

3.2.2 Vytvorenie internetového soketu

```
int socket(int domena, int typ, int protokol);
```

Aplikácia(serverovká aj klientska) si soket vytvára pomocou systémového volania select. Pri vhodnom zvolení atribútov môžmee tento soket nazývať internetový soket. Volanie socket pozostáva z troch atribútov:

- **domena** - špecifikuje sieťové médium používané v rámci komunikácie. My chceme ako médium používať internet, takže domena bude typu AF_INET.

- **typ** - predstavuje spôsob komunikácie, akým sa bude komunikovať po sieťovom médiu. Internet ponúka dva spôsoby komunikácie: pomocou datagramov a prúdov(angl. streams). Prúdové sokety(typ = SOCK_STREAM) medzi sebou vytvárajú akúsi rúru zabezpečujúcu spoľahlivý prenos bytov. Čo sa dostane dnu, vyjde také isté a v rovnakom pradí von. Datagramové sokety(typ = SOCK_DGRAM) oproti prúdovým soketom nezaručujú spoľahlivý prenos bytov, ktoré v rovnakom poradí ako boli odoslané musia prísť, ale za to ponúkajú rýchlejší prenos dát, čo sa dá využiť pri prenose hlasu alebo vysielania. My potrebujeme spoľahlivý prenos údajov, a preto MDemon používa typ SOCK_STREAM.

- **protokol** - využíva sa ak podkladový prenosový mechanizmus umožňuje pre zaistenie príslušného typu viac než jeden prenosový protokol. V prípade internetu sa nič viac nevyužíva, takže sa tam píše 0.

Pri úspešnom volaní, socket vráti súborový deskriptor pre vytvorený soket alebo -1 v prípade zlyhania. Prostredníctvom tohto súborového deskriptoru

sa dá pomocou vytvoreného a pripojeného soketu komunikovať s druhou stranou rovnako ako s otvoreným znakovým zariadením. Príklad vytvorenia internetového soketu:

```
int fd = socket(AF_INET, SOCK_STREAM, 0);
```

3.2.3 Adresa internetového soketu

Serverovské sokety na to, aby mohli počúvať príchodzie spojenia, sa viažu s určitou adresou. Na túto adresu sa potom klientske aplikácie snažia pripojiť. V prípade použitia internetových soketov(soketov s domenou AF_INET) musí mať adresa nasledujúcu štruktúru:

```
struct sockaddr_in{
  short int sin_family /*AF_INET*/
  unsigned short int /*číslo portu*/
  struct in_adress sin_addr /*internetová adresa*/
};
```

kde in_adress je definovaná ako:

```
struct in_adress{
  unsigned long int s_addr;
};
```

Internetová adresa je tu vyjadrená 32-bitovým číslom. Na prevod adresy tvaru „12.34.56.78“ na 32-bitové číslo existuje funkcia:

```
unsigned long int inet_addr(char *adress);
```

Čísla adries a portov sa pri soketovom rozhraní posielajú ako binárne čísla. Rôzne počítače ukladajú čísla v rôznom poradí bytov („litle/big endian“ problém). Kvôli tomuto problému sa dohodol tzv. sieťový formát adries, pričom prevod medzi sieťovým formátom adresy a číslom v počítači umožňujú nasledovné funkcie definované v hlavičkovom súbore netinet/in.h:

- unsigned long int htonl(unsigned long int hostlong); - prevádza číselne vyjadrenú IP adresu na IP adresu v sieťovom tvare.
- unsigned short int htons(unsigned short int hostshort); - prevádza číslo portu na číslo portu v sieťovom tvare.
- unsigned long int ntohl(unsigned long int); - prevádza IP adresu adresu v sieťovom tvare na IP adresu číselne vyjadrenú miestnym počítačom

- unsigned short int ntohs(sunsigned short int); - prevádza číslo portu v sieťovom tvare na číslo portu vyjadrené miestnym počítačom.

Funkcia inet_addr, vracia adresu v sieťovom tvare.

3.2.4 Nastavenie soketu servera aby prijímal spojenia

Aby sa ostatné aplikácie mohli pripájať na soket servera, musí server svoj soket zviazať s nejakou adresou na ktorej bude počúvať pomocou funkcie bind:

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

Bind ako svoje parametre používa súborový deskriptor (sockfd), internetovú adresu v správnom tvare (my_addr) a veľkosť internetovej adresy v bytoch.

Po zviazaní soketu s internetovou adresou, môže soket začať prijímať spojenia pomocu funkcie listen:

```
int listen(int sockfd, int backlog);
```

Listen používa dva parametre. Prvý predstavuje súborový deskriptor soketu, na ktorom počúvame a druhý maximálny počet klientov, ktorý môže čakať vo fronte na vybavenie.

Server vybavuje klientov čakajúcich v rade pomocou volania accept:

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Accept ako svoje parametre používa súborový deskriptor (sockfd), adresu typu sockaddr(addr), kam uloží adresu klienta a adresu, kam uloží veľkosť adresy klienta. Funkcia accept vracia súborový deskriptor priradený k novovzniknutému spojeniu alebo -1 v prípade, že nastala chyba.

3.2.5 Pripojenie soketu klienta k soketu servera

Klientská aplikácia sa pripája na na soket servera pomocou funckie connect:

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

Funkcia connect pripojí svoj soket(sockfd) na adresu servera(serv_addr), s veľkosťou adresy v bajtoch(addrlen). Ak sa podarí pripojiť sa na server, connect vráti 0, inak -1.

3.3 Komunikácia medzi klientskou aplikáciou a programom MDemon

Rovnako ako pri komunikácií medzi modemom mobilu a MDemonom, aj pri komunikácií medzi MDemonom a klientskými aplikáciami si treba dohodnúť isté pravidlá komunikácie. Zatiaľ, čo v prípade komunikácie s modemom mobilu tu existujú medzinárodné štandardy, ktoré treba dodržiavať, na komunikáciu s klientskými aplikáciami si môžeme(resp. musíme) vymyslieť vlastné. Výmena informácií medzi programom MDemon a pripojenou klientskou aplikáciou prebieha pomocou vzájomnej výmeny textových správ ukončených znakom konca riadku<LF>.

Klientské aplikácie posielajú programu MDemon príkazy s parametrami v nasledujúcom tvare podľa počtu parametrov:

```
PRÍKAZ arg1 arg2 ... argN<LF>
```

MDemon daný príkaz spracuje a späť pošle odpoveď v tvare:

```
<číslo odpovede>:Správa popisujúca číslo odpovede<LF>
```

<číslo odpovede> sa nahradí celým číslom v desiatkovom tvare v intervale od 0 do 999, nasleduje dvojbodka, za ktorou nasleduje verbálne vyjadrenie odpovede ukončené znakom konca riadka. Program MDemon pozná príkazy AUTH, SUSPEND, RESUME, SENDSMS a CLOSE. Ak klient pošle príkaz, ktorý MDemon nevie spracovať, MDemon mu späť vráti nasledujúcu správu:

```
400:neviem spracovať danú správu<LF>
```

3.3.1 AUTH login heslo

Príkaz slúži na autorizáciu klienta pomocou prihlasovacieho mena(login) a hesla zadávaných v uvedenom poradí. Login aj heslo pozostáva z 1 až 255 tlačiteľných znakov dolnej časti ASCII tabuľky(znaky 0 až 127) okrem znakov patriacich do kategórie tzv. „white-spaces“, kam patria znaky ako enter, medzera, tabuláto, koniec riadka,... Príkaz slúži na zistenie, či má klientská aplikácia oprávnenie ovládať mobil a mal by tvoriť prvý príkaz klientskej aplikácie. Neautorizovaný klient nemá právo program MDemon odpojiť od mobilu, pripojiť MDemon k modemu mobilu ani poslať SMS správu. Po úspešnej autorizácií, ostáva klient autorizovaný a nemusí sa znovu autorizovať pokiaľ sa nepreruší spojenie medzi klientskou aplikáciou a programom MDemon alebo aplikácia nepošle znovu príkaz AUTH s nesprávnymi argumentami.
MDemon v prípade úspešnej autorizácie pošle nasledujúcu správu:

```
200:AUTH succeeded<LF>
```

V prípade neúspechu pošle jednu z nasledujúcich správ[2]:

```
406:AUTH failed - zly login alebo heslo<LF>
```

```
400:AUTH failed - zly format vstupu<LF>
```

3.3.2 SUSPEND

Príkaz slúži na odpojenie programu MDemon od modemu mobilu. V prípade úspešného odpojenia vráti MDemon:

```
200:SUSPEND succeeded<LF>
```

V prípade neúspechu pošle jednu z nasledujúcich správ[3]:

```
401:SUSPEND failed - musite sa najprv autorizovat<LF>
```

```
403:SUSPEND failed - MDemon bol suspendovany inym klientom<LF>
```

```
500:SUSPEND failed - neznama chyba<LF>
```

3.3.3 RESUME

Príkaz slúži na opätovné pripojenie programu MDemon k mobilu. V prípade úspešného pripojenia alebo v prípade, že MDemon bol pripojený v čase prijatia príkazu, MDemon pošle:

```
200:RESUME succeeded<LF>
```

V prípade neúspechu pošle jednu z nasledujúcich správ[4]:

```
401:RESUME failed - musite sa najprv autorizovat<LF>
```

```
403:RESUME failed - MDemon bol suspendovany inym klientom,
ktory je stale aktivny<LF>
```

```
500:RESUME failed - neznama chyba<LF>
```

[2]Myslím, že samotný popis v správe je dostatočne výsvetľujúci.
[3]Myslím, že samotný popis v správe je dostatočne výsvetľujúci.
[4]Myslím, že samotný popis v správe je dostatočne výsvetľujúci.

3.3.4 CLOSE

Príkaz slúži na ukončenie programu MDemon. V prípade, že MDemon akceptoval volanie, pošle späť:

```
200:CLOSE succeeded<LF>
```

V prípade, že klient nie je autoriozvaný MDemon pošle:

```
401:CLOSE failed - musite sa najprv autorizovat<LF>
```

3.3.5 SENDSMS cislo text

Príkaz slúži na odoslanie SMS správy s textom (text) na číslo(cislo). Odoslať možno aj prázdnu sms správu umiestnením znaku konca riadka za číslo, kam sa má správa poslať.

Číslo musí byť v medzinárodnom tvare. Jeho začiatok pozostáva zo znaku +, za ktorým nasledujú číslice predstavujúce telefónne číslo. Telefónne číslo nesmie v sebe obsahovať pomocné znaky ako medzera alebo lomítko. Príklad správneho tvaru telefónneho čísla:

```
+421908123456
```

Príklady nesprávne zadaného telefónneho čísla:

```
0908123456
+421 0 908 123 456
0908/123456
```

Za telefónnym číslom nasleduje text sms správy. SMS správa nesmie obsahovať viac ako 160 znakov GSM 7 abecedy. Pri posielaní sms správy sa text správy prekódováva z ascii abecedy do GSM 7 formátu. Niektoré znaky považuje tento formát za špeciálne a kóduje ich pomocou dvoch GSM-7 znakov[5]. Pre jednoduchšiu implementáciu text sms správy nesmie v sebe obsahovať znak konca riadku <LF>. Znak konca riadku sa používa pri komunikácií s programom MDemon ako znak ukončujúci príkaz. SMS správa, ktorá v sebe obsahuje koniec riadku, tak bude odoslaná po koniec riadku, pričom zvyšok správy sa spracuje ako chybný vstup zo strany klienta.
V prípade úspešného odoslania sms správy, MDemon pošle:

```
200:SENDSMS succeeded<LF>
```

[5]Tabuľku GSM 7 znakov spolu s ich kódovaním možno nájsť aj na http://www.developershome.com/sms/gsmAlphabet.asp

V prípade neúspechu pošle jednu z nasledujúcich správ[6]:

```
400:SENDSMS failed - zly format vstupu<LF>

401:SENDSMS failed - musite sa najprv autorizovat<LF>

402:SENDSMS failed - zly format vstupu: sms nesmie
 byt dlhsia ako 160 znakov v gsm7 formate<LF>

403:SENDSMS failed - MDemon je suspendovany<LF>

500:SENDSMS failed - neznama chyba<LF>
```

3.3.6 Bezpečnosť komunikácie

Komunikácia medzi klientskou aplikáciou a programom MDemon nie je nijako šifrovaná a všetky informácie sa vymieňajú také aké sú vrátane hesiel. Ak by mal takto komunikovať MDemon s inými aplikáciami cez sieť, takýto spôsob nechránenej komunikácie by predstavoval vážnu bezpečnostnú hrozbu.

Existuje tu však riešenie, ako bezpečnosť komunikácie zvýšiť bez zásahu do kódov programov klientskych aplikácií a programu MDemon. V tomto riešení sa vytvorí akýsi bezpečný tunel, cez ktorý sa komunikuje šifrovane pomocou ssl. Na strane klienta nám stačí presmerovať výstup klienta smerom do tunela, kde sa správa zašifruje a pošle na opačný koniec tunela. Na strane, kde beží MDemon, budeme správy z tunela dešifrovať a posielať na vstup programu MDemon(port, kde bežne MDemon počúva). Odpoveď od programu MDemon sa najprv pošle do tunela, kde sa zase zašifruje a pošle klientovi, ktorý ju na konci dešifruje a pošle klienstkej aplikácií. Toto riešenie sa dá aplikovať využitím rôznych nástrojov. Jeden z takýchto nástrojov môže predstavovať aj program Stunnel využívajúci knižnice Openssl.

3.4 Správa pripojení

Bežiaci MDemon pripojený k modemu mobilu musí priebežne zvládať obsluhovať správy od mobilného telefónu, pripájať k sebe klientske aplikácie a obsluhovať požiadavky pripojených klientskych aplikácií. K tomuto účelu využíva systémové volanie select.

[6]Myslím, že samotný popis v správe je dostatočne výsvetľujúci.

```
int select(int n,
           fd_set *readfds,
           fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);
```

Funkcia select pracuje so súborovými deskriptormi sledovaných zariadení a soketov. Tieto súborové deskriptory má uložené v troch množinách typy fd_set. Prvá množina(readfds) v sebe obsahuje deskriptory súborov sledovaných na čítanie. Druhá množina(writefds) pozostáva z deskriptorov súborov sledovaných pre zápis. Sem patria zariadenia, na ktoré chce program zapisovať, ale nemusia byť momentálne voľné pre zápis. Operáciu write vykoná až keď sa zariadenie uvolní pre zápis. Poslednú množinu tvoria deskriptory s chybovým stavom. S množinami typu fd_set sa pracuje pomocou funkcí:

```
void FD_ZERO(fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
```

Funkcia FD_ZERO slúži na vyprázdnenie množiny súborových deskriptorov(fdset), pomocou FD_SET sa pridáva súborý deskriptor(fd) do určenej množiny súborových deskriptorov(fdset), FD_CLR vymaže deskriptor súboru(fd) z množiny deskriptorov(fdset) a FD_ISSET slúži na testovanie, či sa daný súborový deskriptor(fd) nachádza v množine deskriptorov(fdset).

Funkcia select sa volá s nasledujúcimi argumentami: Prvý argument predstavuje číslo(n) najvyššieho súborového deskriptoru, pokiaľ má sledovať súborové deskriptory. Funkcia select sleduje zariadenia s deskriptormi súborov v intervale od 0 po n-1. Ak chceme sledovať všetky zariadenia, za n treba dosadiť hodnotu najvyššieho deskriptor súboru + 1. Za týmto číslom nasleduje množina deskriptorov sledovaných pre čítanie, množina súborových deskriptorov sledovaných na zapisovanie a množina zapisovačov, kde sa majú objaviť zariadenia s chybou. Ak nás nejaká z množín nezaujíma, resp. by ostala prázdna, možno namiesto nej napísať NULL. Posledný argument predstavuje maximálny čas, ktorý má funkcia select čakať kým sa neozve nejaké zariadenie že má niečo na čítanie, je voľné na zapisovanie, alebo je v chybovom stave. Ak sa sem napíše 0, select bude čakať hoci aj donekonečna, kým sa nebude dať niečo robiť s nejakým sledovaným zariadením.

Volanie select končí ak je nejaké zo sledovaných zariadení pripravené na čítanie, zápis, alebo obsahuje chybový stav, alebo vypršal maximálny časový limit. Volanie select v sebe manipuluje s množinami súborových deskriptorov tak, že po skončení volania množina deskriptorov sledovaných pre čítanie

obsahuje len deskriptory pripravené na čítanie, množina deskriptorov sledovaných pre zapisovanie obsahuje len deskriptory zariadení pripravených na zápis a množina sledujúca chybové hlásenia obsahuje súborové deskriptory, kde vznikol chybový stav. Volanie select vracia počet deskriptorov v jeho troch množinách alebo 0 v prípade, že vypršal časový limit. Ak nastala nejaká chyba, select vráti -1.

3.4.1 Funkcia pocuvaj

Sledovanie mobilu, klientov a voľného serverovského soketu spolu so spracovávaním výsledkov a požiadaviek sledovaných zariadení má v programe MDemon na starosti jeho vlastná funkcia pocuvaj:

```
int pocuvaj();
```

Funkcia pocuvaj sleduje prostredníctvom select len množinu deskriptorov čakajúcich na čítanie. Sem patria súborové deskriptory klientskych aplikácií, sériový port pripojeného mobilu a voľný soket, čakajúci na klientské sokety. Pri volaní select funkcia pocuvaj nastavuje časový limit na 0, takže čaká až kým aspoň jedno zo sledovaných zariadení a soketov nemá niečo na čítanie, alebo voľný serverovský soket nemá klientsky soket, ktorý by sa chcel naň pripojiť.

Táto funkcia končí len v prípade, že nastala chyba, niekto zrušil a ukončil bežiaci proces predstavujúci program MDemon, alebo autorizovaná klientská aplikácia poslala CLOSE. V prípade chyby vráti -1. Za chybu sa považuje aj ak sa mobil z neznámych príčin sám odpojil od programu MDemon. Ak sa nejaké zariadenie odpojí od programu, pošle mu 0 znakov. Takto sa dá zistiť, či sa napríklad sám odpojil modem, alebo nejaký pripojený klient.

Funkcia pocuvaj vo svojej podstate pracuje v nekonečnom cykle, v ktorom si nastaví množinu zariadení, ktoré chce počúvať a zavolá na ne select. V prípade, že bol program MDemon odpojený nejakou klientskou aplikáciou(bol suspendovaný), množina sledovaných zariadení neobsahuje súborový deskriptor sériového portu mobilu a zaradí sa tam, až bude toto spojenie obnovené.

Ak volanie select skončí s chybou(vráti -1), funkcia počúvaj skončí tiež s chybou -1. V prípade normálneho skončenia funkcie select začne funkcia zisťovať, ktorý so sledovaných deskriptorov sa nachádza v množine deskriptorov sledovaných na čítanie. To robí testovaním rad za radom výskytu počúvaných zariadení v množine zariadení pripravených na čítanie resp. obslúženie. Pripravené zariedenie, potom náležite obslúži a pokračuje v ďalších zariadeniach. V jednom momente program MDemon obsluhuje len jedno zariadenie. Pre potreby tohto programu to stačí.

Ako prvý testuje, či neprišla správa od mobilu. Ak prišla zistí, koľko znakov obsahuje správa. Ak mobil poslal 0 znakov, funkcia vracia -1 a končí. Inak zavolá funkciu obsluz_mobil s parametrom vyjadrujúcim počet znakov, ktoré má mať správa. Funkcia obsluz_mobil potom prečíta správu od mobilu a pokúsi sa ju ďalej spracovať. Ak prečíta menej znakov ako má skončí s -1 a následne aj funkcia pocuvaj skončí rovnako, pretože mobil sa samovoľne odpojil od programu MDemon.

Za mobilom, testuje klientské aplikácie, či neprišla od niektorej nejaká požiadavka. Ak áno tak ju vybaví. Ak nejaká poslala 0 znakov, odpojí ju od programu MDemon a vymaže zo zoznamu pripojených klientov. Ak poslala viac ako 0 znakov, pocuvaj zavola funkciu spracuj_klienta, ktorá prečíta správu od klienta a následne ju spracuje. Ak počas tohto volania vysvitne, že sa MDemon samovoľne odpojil od mobilu(napríklad po poslaní príkazu na odoslanie sms mobilu, mobil pošle 0 znakov), spracuj klienta vráti -1 a pocuvaj následne tiež.

Posledným testovaným je voľný soket čakajúci na klientov. Ak sa objavil klient, pocuvaj pomocou volania accept prijíme klienta a pridá ho do zoznamu pripojených klientov.

3.5 Konfigurácia programu MDemon

Program MDemon na svoj chod potrebuje poznať niekoľko hodnôt a nastavení. Potrebuje vedieť cestu k sériovému portu mobilu, poznať internetovú adresu, s ktorou má zviazať svoj soket na prijímanie klientskych aplikácií, poznať telefónne čísla a programy, ktoré má spustiť, keď zavolá na mobil dané telefónne číslo a v neposlednom rade musí vedieť prihlasovacie mená a heslá aplikácií oprávnených ovládať program MDemon, aby ich mohol autentifikovať.

Nastavenia pre program MDemon sa nachádzajú v súboroch mdemon.conf a mdemonusers.conf. Po bežnej inštalácií by sa tieto súbory mali nachádzať v priečinku /usr/local/etc/mdemon.

3.5.1 mdemon.conf

Súbor mdemon.conf predstavuje obyčajný textový súbor, v ktorom by sa mali nachádzať nasledovné informácie v presne tomto poradí:

- prístupová cesta k súboru predstavujúcom sériový port mobilu
- internetová adresa pozostávajúca z IP adresy a čísla portu, kam sa majú pripájať klientske aplikácie

- telefónne čísla a programy, ktoré má spustiť MDemon, keď zavolá na mobil dané telefónne číslo

Súbor mdemon.conf pozostáva z riadkov nastavení, prázdnych riadkov a riadkov slúžiacich ako komentár. Riadok nesmie obsahovať viac ako 1000 znakov. Ak nejaký riadok bude dlhší ako 1000 znakov, program MDemon vyhlási chybu a nebude tento riadok ďalej spracúvať.

Riadky slúžiace ako komentáre začínajú znakom mriežky '#'. Ak riadok nebude začínať znakom mriežky, bude vnímaný ako prázdny alebo ako riadok s nastavením. Príklady riadku predstavujúci komentár, prázdny riadok a chybný riadok, ktorý chcel byť komentár, ale nie je:

```
# Toto je riadok komentáru

 # toto je chybný riadok, ktorý nie je komentár
```

Riadky s nastavením často začínajú s príkazom, definujúcim, čo sa má nastaviť a parametrom nastavenia. Výnimku predstavuje časť súboru slúžiaca na načítanie telefónnych čísel spolu s programmi, ktoré má spustiť MDemon, keď zavolá na mobil dané telefónne číslo.

Prístupová cesta k sériovému portu mobilu sa zadáva do riadku, vyhradeného len pre toto nastavenie, pomocou príkazu DEVICE:

```
DEVICE <cesta>
```

Za príkazom DEVICE nasleduje parameter <cesta>, ktorý sa nahradí reťazcom predstavujúcim cestu k sériovému portu mobilu.

Internetová adresa, kde má počúvať soket programu MDemon klientske aplikácie, sa nastavuje pomocou príkazov IP a PORT:

```
IP <ip adresa>
PORT <číslo portu>
```

Každý z týchto príkazov sa zadáva do riadku, vyhradeného len pre tento príkaz s parametrom. V prípade nastavenia IP, <ip adresa> sa nahrádza textovým vyjadrením IP adresy, alebo slovom ALL, pomocou ktorého sa nastavuje, že soket programu MDemon má počúvať na všetkých adresách, ktoré počítač, na ktorom beží MDemon vlastní. Pomocou príkazu PORT sa nastavuje číslo portu, na ktorom má MDemon počúvať. <číslo portu> sa tu nahradí číslom portu v desiatkovej sústave. Príklad nastavenia internetovej adresy, kde má MDemon počúvať na ip adrese 127.0.0.1 a porte 12345:

```
IP 127.0.0.1
PORT 12345
```

Poslednú časť programu predstavuje časť obsahujúca telefónne čísla a programy, ktoré má spustiť MDemon, keď zavolá na mobil dané telefónne číslo. V prípade, že táto časť bude obsahovať viac rovnakých telefónny čísel s rôznymi programmi, MDemon po zavolaní na takéto číslo spustí prvý z uvedených programov. Táto časť sa začína riadkom obsahujúcim len slovo CALLS. Za týmto riadkom nasledujú riadky v takomto formáte:

```
<tel.číslo> <program>
```

<tel.číslo> sa nahradí telefónne číslo, ktoré musí byť napísané v celku, bez medzier a pomocných znakov, vyjadrené v desiatkovomm tvare. <program> sa nahradí volaním programu s argumentami vrátane nultého argumentu, ktorý väčšinou predstavuje názov programu.

Príklad konfiguračného súboru mdemon.conf:

```
# Configuration file for mdemon

# serial port adress of GSM/GPRS device
device /dev/rfcomm0

# ip adress and port, where mdemon is listening to clients
# ALL indicates, that mdemon is listening at all IP adresses
IP ALL
PORT 9765

CALLS
0949112233 /home/danica/bakalarka/demon/klient klient
0949112234 /home/danica/bakalarka/mdemon/program program arg1 arg2
```

3.5.2 mdemonusers.conf

Súbor mdemonusers.conf predstavuje obyčajný textový súbor, ktorý v sebe obsahuje prihlasovacie mená a heslá aplikácií oprávnených manipulovať s programom MDemon. Tento súbor môže pozostávať z prázdnych riadkov, riadkov tvoriacich komentár a riadkov obsahujúcich prihlasovacie mená a heslá užívateľov aplikácií oprávnených manipulovať s programom MDemon. Riadok nesmie obsahovať viac ako 1000 riadkov. Riadok prevyšujúci 1000 znakov bude vyhlásený za chybný a nebude sa ďalej spracúvať.

Riadky slúžiace ako komentáre začínajú znakom mriežky '#'. Ak riadok nebude začínať znakom mriežky, bude vnímaný ako prázdny alebo ako riadok s nastavením. Príklady riadku predstavujúci komentár, prázdny riadok a chybný riadok, ktorý chcel byť komentár, ale nie je:

```
# Toto je riadok komentáru

 # toto je chybný riadok, ktorý nie je komentár
```

Každý oprávnený užívateľ má svoje prihlasovacie meno a heslo uložené v riadku, ktorý môže obsahovať len toto prihlasovacie meno a heslo. Riadok má nasledovný formát:

```
<login> <heslo>
```

<login> sa nahradí prihlasovacím menom užívateľa a <heslo> sa nahradí jeho heslom. Prihlasovacie meno a heslo musia mať počet znakov v intervale 1 až 255, pričom nesmú obsahovať tzv. „white-spaces“.

Príklad konfiguračného súboru mdemonusers.conf:

```
#List of authorized users, that can manipulate with mdemon. Each line
# represents one's user login and password necessary for authentification
# to mdemon
user user123
mrkvicka mrkvicka123
```

Kapitola 4

Vzdialený prístup pre administrátora

Táto kapitola sa zaoberá vytvorením vzdialeného prístupu pre administrátora v prípade nedostupnosti primárnych sieťových kanálov. Tento vzdialený prístup sa uskutoční pomocou pripojenia sa k sieti internet za asistencie mobilného telefónu, ktorý sa využije ako dial-up modem. Pripojením počítača k internetu cez modem mobilu sa zaoberá prvá časť tejto kapitoly.

Po pripojení sa počítača k internetu cez modem mobilu však stále nebudeme schopní sa naň pripojiť zo siete internet, pretože poskytovatelia internetových služieb cez mobil neprideľujú zariadeniam verejné IP adresy. Pripojiť k počítaču sa môžeme jedine tak, že počítač pripojíme k inému počítaču s verejnou IP adresou. V tejto využijeme program OpenVPN, pomocou ktorého prepojíme počítač s neverejnou IP adresou s počítačom s verejnou IP adresou, tak že si budú navzájom tvoriť kvázi lokálnu sieť a budú adresovateľné ako počítače v lokálnej sieti.

Keď počítač bude adresovateľný druhému počítaču ako počítač v lokálnej sieti, druhý počítač bude môcť využívať jeho služby a pripojiť sa tak k nemu napríklad pomocou ssh a vzdialene sa prihlásiť ako užívateľ systému.

4.1 Pripojenie k internetu pomocou mobilu

Mobilným telefónom umožňuje prístup k internetu ich GPRS modem. S modemamy využívajúce novšie technológie ako napríklad EGPRS a 3G sa pracuje rovnako ako s GPRS modemami. Modem umožňuje týmto telefónom tzv. dial-up pripojenie, pracujúce na podobnom princípe ako dial-up pripojenie cez telefónnu linku. Pripojením počítača k modemu mobilu, sme schopní tento modem využívať na pripojenie sa počítaču do siete internet.

Na pripojenie počítača k internetu pomocou dial-up modemu slúži PPP protokol. PPP(Point-to-Point Protocol) operuje na dátovej vrstve a slúži na vytvorenie priameho spojenia medzi dvoma uzlami sieťovej komunikácie. Môže zabezpečiť autentifikáciu, kompresiu a kódovanie prenášaných údajov. Mobilný operátori poskytujúci internet cez mobil, využívajú práve tento PPP protokol. Na pripojenie sa do internetu treba najprv inicializovať modem a následne vytočiť číslo dané poskytovateľom internetu. Niektorí poskytovatelia vyžadajú ešte autentifikáciu prihlasovacím menom a heslom. Keď všetko prebehne ako má, poskytovateľ internetových služieb pripojí počítač(mobil) do siete internet. Nevýhodou týchto pripojení je, že poskytovaľ internetových služieb väčšinou nepridelí takto pripojenému zariadeniu verejnú IP adresu. To má za následok, že sa ostatné počítače v sieti nemôžu na dané zariadenie pripojiť.

V Linuxe sa na pripojenie k sieti internet pomocou dial-up modemu dá použiť niekoľko nástrojov. Pre svoju jednoduchosť a možnú použiteľnosť vo viacerých Linuxových distribúciach bez viazanosti na konkrétne grafické rozhranie, som sa rozhodla použiť program WvDial.

WvDial je program, ktorého úlohou je vytočiť modem a pripojiť počítač k internetu pomocou PPP. Po spustení program načíta svoj konfiguračný súbor z /etc/wvdial.conf slúžiaci na nastavenie pripojenia. Konfiguračný súbor obsahuje základné informácie o umiestnení modemu, jeho rýchlosti, informáciach o poskytovateľovi internetových služieb(telefónne číslo, prihlaosvacie meno, heslo) a inicializačné reťazce, ktorými má byť modem nastavený. Na základe týchto informácií inicializuje(nastaví) modem a vytočí číslo pre pripojenie sa do internetu. Po pripojení sa do internetu zavolá program pppd a nechá ho dokončiť zvyšnú robotu. Výsledkom úspešného volania by mal byť počítač pripojený pomocou PPP do internetu. Pre bližšie informácie odporúčam manuálové stránky wvdial, wvdial.conf a pppd.

Príklad konfiguračného súboru pre pripojenie sa do internetu pomocou modemu mobilu umiestneného na /dev/rfcomm0 pripojeného k sieti O2 Slovensko:

```
[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","o2internet"
Baud = 460800
New PPPD = yes
Modem = /dev/rfcomm0
ISDN = 0
Phone = *99#
```

```
Password = ''
Username = ''
Stupid Mode = yes
```

Init1, Init2 a Init3 predstavujú inicializačné reťazce, ktrými sa modem nastavuje, aby ho bolo možné pripojiť k internetu. Prvé 2 sa používajú pre väčšinu dial-up modemov. Posledný reťazec(Init3) je typický pre GPRS a im podobné modemy a slúži na (pre)definovanie profilu pripojenia na IP s menom prípojného miesta(Access Point Name) daného poskytovateľom služieb. Baud vyjadruje maximálny prenosovú rýchlosť modemu a Phone, číslo, ktoré sa má vytočiť. New PPPD sa musí označiť za yes v prípade, že počítač využíva k pripojeniu pppd verzie 2.3.0 a novšie. Stupid mode označuje, že po pripojení modemu sa má hneď zavolať pppd. Keďže O2 internet cez mobil na Slovensku nevyžaduje autorizáciu pomocou prihlasovacieho mena a hesla, definujú sa ich hodnoty pomocu dvoch apostrofov symbolizujúcich prázdny reťazec.

Príklad úspešného volania programu WvDial spolu s jeho výstupom na terminál.

```
#wvdial
--> WvDial: Internet dialer version 1.60
--> Cannot get information for serial port.
--> Initializing modem.
--> Sending: ATZ
ATZ
OK
--> Sending: ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
OK
--> Sending: AT+CGDCONT=1,"IP","o2internet"
AT+CGDCONT=1,"IP","o2internet"
OK
--> Modem initialized.
--> Sending: ATDT*99#
--> Waiting for carrier.
ATDT*99#
CONNECT
~[7f]}#@!}!} } }2}#}$@#}!}$}%\}"}&} }*} } g}%~
--> Carrier detected.  Starting PPP immediately.
--> Starting pppd at Tue Jun  1 12:47:56 2010
--> Pid of pppd: 5718
--> Using interface ppp0
```

```
--> pppd: (E[07] @C[07]  G[07]
--> pppd: (E[07] @C[07]  G[07]
--> pppd: (E[07] @C[07]  G[07]
--> pppd: (E[07] @C[07]  G[07]
--> pppd: (E[07] @C[07]  G[07]
--> local  IP address 91.191.92.220
--> pppd: (E[07] @C[07]  G[07]
--> remote IP address 10.6.6.6
--> pppd: (E[07] @C[07]  G[07]
--> primary   DNS address 160.218.43.200
--> pppd: (E[07] @C[07]  G[07]
--> secondary DNS address 160.218.10.200
--> pppd: (E[07] @C[07]  G[07]
```

4.2 OpenVPN

OpenVPN je opensource aplikácia implementujúca VPN(Virtual Private Network). OpenVPN umožňuje prepájať viacero vzdialených počítačov do jednej virtuálnej siete, kde medzi sebou môžu komunikovať akoby by boli pripojené pomocou LAN. Navyše OpenVPN zabezpečuje aj bezpečnosť takéhoto prepojenia a snaží sa zabezpečiť dôvernosť, integritu a autentickosť prenášaných informácií v rámci svojej virtuálnej siete pomocou hesiel, klúčov, či certifikátov. Dva vzdialené počítače sa prepoja pomocou OpenVPN tak, že medzi sebou vyvtoria akýsi zabezpečený tunel, cez ktorý posielajú všetku vzájomnú komunikáciu. Výhodou OpenVPN je, že umožňuje vzdialené pripojenie sa do siete aj počítačom, skrytými za NATs(nemajú verejnú IP adresu) a firewallom.

V našom prípade máme počítač pripojený k internetu bez verjnej IP adresy. Pomocou OpenVPN ho môžme pripojiť k nejakému inému počítaču s verejnou ip adresou a vytvoriť tak malú virtuálnu sieť medzi týmito dvoma počítačmi. Počítač s verejnou IP adresou tu bude vystupovať ako server, na ktorý sa pripojí klient predstavujúci náš počítač. Pre naše účely stačí ak medzi nimi vznikne point-to-point prepojenie.

Prvým predpokladom je mať na oboch stranách nainštalované OpenVPN. Druhý krok predstavuje správna konfigurácia klienta aj servera tak, aby medzi sebou mohli komunikovať. Dôležitú časť konfigurácie zabezpečuje spôsob preukázania autenticity na oboch stranách. Pre tento účel sa dá použiť overovanie pomocou certifikátov, alebo akýsi kľúč predstavujúci zdieľané tajomstvo. Pre svoju jednoduchosť a názornosť použijem zabezpečenie pomocou kľúča predstavujúceho zdielané tajomstvo.

Kľúč predstavujúci zdieľané tajomstvo sa vytvorí ľahko v konzole pomocou príkazu:

```
openvpn --genkey --secret static.key
```

Vytvorený kľuč - static.key musí obsahovať server aj klient. Dajme tomu aby sa static.key vyskytoval na oboch stranách v adresári /etc/openvpn. Kľúč by mal byť tajný širokému okoliu, preto by bolo vhodné mu nastaviť práva tak aby ho nik okrem administrátora nesmel čítať a ani s ním nijako manipulovať.

Ďalším krokom je vytvorenie konfiguračného súboru pre klienta aj server. Konfiguračný súbor pre klienta sa pritom bude vyskytovať na strane klienta a konfiguračný súbor pre server sa bude nachádzať na strane servera. Príklad konfiguračného súboru s názvom server.conf pre server:

```
dev tun7
ifconfig 10.7.2.8 10.7.2.9
port 1194
proto udp
secret /etc/openvpn/static.key
comp-lzo
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

Počítač sa pripája k sieti pomocou sieťových interface. OpenVPN používa vlastný sieťový interface, ktorý sa v tomto prípade bude volať tun7. Cez tento interface bude prechádzať komunikácia adresovaná nášmu klientskému zariadeniu. Príkaz ifconfig nastavuje serveru pre komunikáciu cez interface tun7 adresu 10.8.7.2.8, pričom klientska aplikácia bude mať adresu 10.7.2.9. Prihádzajúce spojenia od klientskej aplikácie má počúvať na porte 1194, pričom ako komunikačný protokol sa má používať UDP[1]. Port 1194 je východzí port určený pre OpenVPN používajúce službu UDP Príkaz secret nastavuje, že na zabezpečenie komunikácie sa má používať zdielané tajomstvo uložené v /etc/openvpn/static.key. Comp-lzo sa stará o komprimáciu prenášaných údajov. Keepalive zabezpečuje, aby prepojenie vydržalo aj keď sú práve obe strany neaktívne(firewally zviknú niekedy takéto spojenie prerušiť) posielaním ping každých 10 sekúnd. Ak nepríde odozva na ping do 60 sekúnd, server bude vnímať, že nastalo prerušenie spojenia medzi ním a klientom.

[1]TCP by spôsobovalo zbytočné zdržovanie komunikácie dvojitým overovaním, pretože o TCP sa budú aj tak starať aplikácie, ktoré na svoju komunikáciu využívajú TCP. Navyše aplikácie, ktoré by chceli používať UDP by používali TCP, čo nechceme.

Persist-tun a persist-key zabezpečia, aby sa pri reštarte nenačítavali informácie(najmä klúč), najmä potom, čo sa znížia oprávnenia aplikácie. Tie sa dajú znížiť pomocou zmeny užívateľa a skupiny na nobody pomocou príkazov user a group.

Konfiguračný súbor klientskej aplikácie vyzerá podobne. Rozdiel je v tom, že klientskej aplikácií treba zadať ešte IP adresu spolu s portom, kde čaká server na pripájajúcich sa klientov(v našom prípade len na jedného). Príklad konfiguračného súboru s názvom klient.conf pre klienta:

```
dev tun7
remote 192.168.64.100 1194
ifconfig 10.7.2.8 10.7.2.9
port 1194
proto udp
secret /etc/openvpn/static.key
comp-lzo
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

Konfiguračné súbory je dobré si uložiť v adresári /etc/openvpn avšak nie je to nevyhnutnosť.

Posledným krokom je vytvorenie samotného spojenia. Najprv treba spustiť OpenVPN na strane servera pomocou príkazu openvpn s parametrom konfiguračného súboru pre server:

```
openvpn /etc/openvpn/server.conf
```

Keď server počúva, klient sa može pripojiť pomocou volania:

```
openvpn /etc/openvpn/klient.conf
```

Ak všetko vyšlo ako malo, mali by sme teraz mať prepojené 2 počítače, ktoré spolu odteraz môžu komunikovať po zabezpečenom kanáli. Klient môže pristupovať k serveru cez IP 10.7.2.8 a čo je pre nás najdôležitejšie, server bude môcť pristupovať k službám klienta na IP 10.7.2.9.

4.3 Vzdialený prístup pomocou SSH

SSH(Secure Shell) je internetový protokol umožňujúci dvom zariadeniam prenos dát cez zabezpečený kanál. SSH bolo vyrobené primárne pre Linuxové

a Unixové systémy umožňujúce vzdialené prihlásenie sa do systému. Jeho cieľom bolo nahradiť Telnet a jemu podobné aplikácie, ktoré posielali všetky dáta vrátane hesiel nezašifrované internetom, ich bezpečnejšou variantou.

SSH sa väčšinou využíva na prihlásenie sa do vzdialeného počítača na spúsťanie shellových príkazov, ale dá sa využiť aj pre tunelovanie a presmerovávanie komunikácie(port forwarding). Autentifikáciu užívateľov zabezpečuje SSH pomocou prihlasovacieho mena a hesla, ale dá sa nastaviť aby totožnosť overoval pomocou verejných a súkromných kľúčov.

Aby sme sa mohli pomocou SSH vzdialene prihlásiť na náš počítač, počítač musí mať nainštalovaný a nakonfigurovaný SSH server a spusteného daemon počúvajúceho na nejakom porte(SSH má pre svoju komunikáciu vyhradený port 22).

Správanie SSH servera, resp. jeho daemona, sa konfiguruje pomocou súboru /etc/ssh/sshd_config.[2] Tu sa dá nastaviť na akom porte má počúvať, spôsob autentifikácie s klientskymi aplikáciami a presmerovávanie portov. Pri ponechaní východzích nastavení sa bude dať prihlásiť do počítača pomocou prihlasovacieho mena a hesla, pričom ssh klientov treba pripájať na port 22.

Príklad vzdialeného sa prihlásenia na náš počítač ako užívateľ virtulko pomocou ssh:

```
# ssh virtulko@10.7.2.9
virtulko@10.7.2.9's password:
Linux virtulko-laptop 2.6.31-14-generic #48-Ubuntu SMP
 Fri Oct 16 14:04:26 UTC 2009 i686

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

264 packages can be updated.
108 updates are security updates.

Last login: Wed Jun  2 15:32:11 2010 from 10.7.2.8
virtulko@virtulko-laptop:~#
```

[2]Pre viac informácií ohľadom nastavení a používania OpenSSH odporúčam si pozrieť ich internetové a manuálové stránky.

Kapitola 5

Aplikácie využívajúce program MDemon

MDemon sa dá ovládať len pomocou ďaľších naprogramovaných aplikácií, ktoré sa pripájajú na program MDemon. Pre zjednodušenie tvorby aplikácií pripájajúcich sa na MDemon, balík s programom MDemon ponúka aj statickú knižnicu mdemonaplib.a uloženú v priečinku mdemonaplib spolu s hlavičkovým súborom mdemonapl.h.

Okrem programu MDemon a knižnice, balík obsahuje aj aplikácie slúžiace na ovládanie programu MDemon z konzoly a 2 programi. Tie 2 programi slúžia ako príklady programov, ktoré by mohli využívať program MDemon pri detekcií straty spojenia do siete internet a vytváraní možnosti vzdialene sa prihlásiť administrátorovi. Posledná kapitola tejto práce je venovaná týmto aplikáciam.

Všetky aplikácie, ktoré chcú ovládať program MDemon sa naň musia najprv pripojiť pomocou internetového soketu a autentifikovať sa. Na pripojenie sa pomocou internetového soketu potrebujú poznať IP adresu a port, kam sa majú pripojiť. IP adresu a port hľadajú priložené aplikácie v súbore mdemonadr.conf štandardne umiestnenom v adresári /usr/local/etc/mdemon. Na autentifikáciu potrebujú aplikácie poznať prihlasovacie meno a heslo užívateľa oprávneného ovládať program MDemon. Prihlasovacie meno a heslo na autentifikáciu hľadajú uvedené programy v /usr/local/etc/mdemon/mdemonusr.conf. Oba konfiguračné súbory sa riadia rovnakými pravidlami ako konfiguračné súbory programu MDemon. Jediný rozdiel je v tom, že mdemonadr.conf obsahuje len príkazy IP a PORT a mdemonusr.conf len jedného oprávneného užívateľa programu MDemon, a to užívateľa, pod ktorého identitou sa majú programy autentifikovať. Časť zaoberajúca sa načítavaním údajov z konfiguračných súborov, pripojením sa na internetový soket programu MDemon a autentifikáciou užívateľa je pre

všetky aplikácie dodávané spolu s programom MDemon rovnaká. Programy na pripojenie sa k programu MDemon a autentifikáciu využívajú funkciu mdconnect_and_auth z knižnice mdemonaplib.a. Tá sa stará o načítanie údajov zo štandardne umiestnených konfiguračných súborov, pripojenie sa k programu MDemon a autentifikáciu užívateľa. V prípade zlyhania vráti 1 a chybu zapíše na stderr. V prípade úspechu vráti súborový deskriptor soketu pripojeného k programu MDemon.
Príklady konfiguračných súborov mdemonadr.conf a mdemonusr.conf:

```
# mdemonadr.conf - Adresa, kde počúva MDemon
IP 127.0.0.1
PORT 9765

# mdemonusr.conf - prihlasovacie údaje užívateľa
# pod ktorým aplikácie na tomto PC pracujú s
# s programom MDemon
# login heslo
user user123
```

5.1 mdsuspend

Program mdsuspend slúži na odpojenie programu MDemon od modemu mobilu. Volá sa bez argumentov a okrem inicializačnej časti, spoločnej s ostatnými programmi, pozostáva z vyslania príkazu SUSPEND programu MDemon a počkania na odpoveď. Odpoveď od programu vypíše na terminál a skončí. Program mdsuspend skončí úspešne vtedy, kedy aj vyslanie príkazu SUSPEND programu MDemon spôsobí odpojenie programu MDemon od mobilu.

5.2 mdresume

Program mdresume sa rovnako ako mdsuspend volá bez argumnetov a slúži na opätovné pripojenie programu MDemon k modemu mobilu. Okrem inicializačnej časti, pozostáva z vyslania príkazu RESUME programu MDemon a počkania na odpoveď. Čakanie na odpoveď môže trvať aj niekoľko sekúnd. To je spôsobené tým, že po obnovení spojenia medzi modemom mobilu a programom MDemon, MDemon znovunastavuje modem mobilu. Kladnú odpoveď pošle až keď je úspešne pripojený s znovunastaveným modemom. Odpoveď od programu MDemon vypíše program mdsuspend na terminál a skončí. V prípade nastania inej chyby, samozrejme vypíše tú a

skončí. Program mdsuspend skončí úspešne vtedy, kedy aj vyslanie príkazu SUSPEND programu MDemon spôsobí odpojenie programu MDemon od mobilu.

5.3 mdclose

Program mdclose sa volá bez argumentov a slúži na ukončenie programu MDemon. Po inicializačnej časti pošle mobilu príkaz CLOSE a jeho odpoveď prepošle na terminál. Príkaz môže skončiť neúspechom ak sa nepodarilo pripojiť a autentifikovať na server, prerušilo sa spojenie medzi mdclose a programom MDemon(ak však MDemon získal správu pred prerušením spojenia, mohol skončiť hoc mdsuspend o tom nemá správu).

5.4 mdsendsms

Program mdsendsms, ako názov napovedá, slúži na odosielanie sms. Volá sa s dvomi argumentami. Prvý argument predstavuje telefónne číslo a druhý(nepovinný) text správy. V prípade argumentov platia rovnaké pravidlá ako pri posielaní príkazu SENDSMS na port programu MDemon. Ak je mdsendsms volaný z konzoly, argument predstavujúci text správy musí byť ohraničený úvodzovkami. Inak, ak bude pozostávať z viac ako jedného slova, budú zvyšné slová považované za ďalšie argumenty programu a program skončí s chybou. Príklad volania programu mdsendsms na odoslanie sms s textom „ahoj bobor“ na číslo 0908123456:

```
mdsendsms +421908123456 "ahoj bobor"
```

Program okrem inicializačnej časti pozostáva z vyslania naraz[1] dvoch príkazov a prijatia 2 odpovedí. Tie môžu aj nemusia prísť naraz. Prvým príkazom je RESUME. V prípade, že je MDemon pripojený k mobilu, alebo sa ho podarilo naozaj pripojiť, vráti MDemon pozitívnu odpoveď. V prípade, že bol MDemon odpojený iným klientom, pošle MDemon späť chybovú správu. Druhým príkazom je už príkaz SENDSMS spolu s argumentami prijatými na vstupe. Po spracovaní druhého príkazu, pošle MDemon zase odpoveď na základe ktorej sa vie, či program uspel alebo zlyhal. Táto odpoveď sa potom interpretuje a v prípade úspechu vráti MDsendsm správu informujúcu o úspešnom odoslaní, alebo skončí s chybovou hláškou.

RESUME a SENDSMS sa posielajú naraz, pretože ak sa ich podarí naraz načítať programu MDemon, MDemon ich spracuje hneď za sebou a zabezpečí

[1] Program MDemon umožňuje posielanie viacerých príkazov naraz.

tak, že medzi volaniami RESUME a SENDSMS nepríde iná aplikácia, ktorá by akurát odpojila MDemon od mobilu.

5.5 mdclose

MDclose sa volá bez argumentov a slúži na skončenie programu MDemon. Okrem inicializačnej časti pozostáva z vyslania príkazu CLOSE programu MDemon a počkania na odpoveď. V Prípade úspechu vypíše program správu o úspechu, inak skončí s chybou.

5.6 mdpripoj

Program mdpripoj na rozdiel od predošlých programov nie je určený na obyčajnú manipuláciu s programom MDemon. Program je myslený ako príklad programu, ktorý by sa mohol spustiť po zavolaní administrátora na mobil. Jeho úlohou je zabezpečiť vzdialený prístup administrátorovi. Ten sa snaží umožniť tak, že po spustení odpojí program MDemon od mobilu, pripojí pomocou programu wvdial počítač na internet cez modem mobilu a pokúsi sa ako klient pripojiť na openvpn server. V prípade neúspechu by sa mal vedieť korektne ukončiť a umožniť programu MDemon znovu počúvať mobil.

Po spustení sa pripája a autentifikuje rovnako ako všetky doteraz spomenuté aplikácie. Po úspešej autentifikácií vyšle programu MDemon príkaz SUSPEND. V prípade úspechu, pokračuje a pomocou systémového volania *fork* sa zduplikuje. K bežiacemu procesu predstavujúci program mdpripoj pribudne jeho druhá súbežne bežiaca kópia. Volanie fork vracia novej kópií číslo 0 a pôvodnému programu číslo jednoznačne identifikujúce novovzniknutý proces medzi ostatnými procesmi bežiacimi v operačnom systéme - *pid*. V prípade chyby vráti volanie fork -1. V takom prípade program mdpripoj pošle programu MDDemon príkaz RESUME a skončí s chybou.

Novovzniknutá kópia programu(nazývaná aj dieťa programu-procesu) spustí pomocou systémového volania *execv* wvdial. V prípde, že sa podarí spustiť volaný program, program preberie číslo procesu(pid) volajúcemu procesu a volajúci proces skončí. V prípade neúspechu volania execv(program sa z nejakých príčin nepodarilo spustiť), execv vráti -1 a skončí s chybovou hláškou. V takomto prípade dieťa mdpripoj stvorené pre zavolanie wvdial končí neúspechom a ostáva bežať len pôvodná(rodičovská) vetva programu.

Úspešne spustený wvdial sa pokúsi pripojiť na modem mobilu, inicializovať ho a spustiť program *pppd*, ktorý sa má postarať o zvyšok, čo sa týka pripojenia na internet cez mobil. Program wvdial spustí pppd ako nový proces.

Program pppd zabezpečí vytvorenie sieťového interface(u), väčšinou ppp0, cez ktorý sa vytvorí pomocou PPP pripojenie počítača do siete internet. Interface ppp0 prevádzkuje proces, ktorého pid je uložený v /var/run/ppp0.pid. Poslaním signálu SIGTERM tomuto procesu, zabezpečíme prerušenie spojenia medzi počítačom a okolitým svetom pomocou ppp0 a uvoĹníme sériový port mobilu.[2]

Vráťme sa teraz k časti, kedy sa mdpripoj úspešne zduplikoval, a zamerajme sa na pôvodnú(rodičovskú) kópiu programu. Jej úlohou je pomocou openvpn pripojiť počítač ako klienta k openvpn serveru. To sa dá až potom, čo sa počítač pripojí k internetu. Pripojenie sa na internet niečo trvá a tak rodičovská kópia, kým niečo začne robiť, zaspí na 10 sekúnd pomocou systémového volania *sleep*. Po krátkom odpočinku sa program zobudí a zduplikuje. Vytvorené dieťa uloží svoj pid(získaný systémovým volaním *getpid)* do /usr/local/etc/mdemon/openvpn.pid a spustí openvpn s príslušnými parametrami pomocou *execv*. Po úspešnom spustení openvpn bude openvpn bežať s rovnakým pid ako bežalo dieťa. Vďaka uloženiu pid, sme schopní hocikedy poslať signál na ukončenie programu openvpn. V prípade, že sa nepodarí spustiť openvpn, dieťa zmaže súbor openvpn.pid. Ak sa bude neskôr nejaká aplikácia pokúšať ukončiť ukončené openvpn pomocou získaného pid z tohto súboru, nepodarí sa jej to, lebo súbor bude zmazaný. Toto slúži ako ochrana pred ukončením nesprávneho procesu omylom.

Nastavenie openvpn spojenia tiež niečo trvá a tak rodičovský proces zas zaspí na 10 sekúnd. Po prebudení si začne overovať, či sa naozaj podarilo spojiť s openvpn serverom. Overovanie prebehne tak, že sa pokúsi cez openvpn tunel poslať na druhú stranu 5 echo-request paketov. Ak do primeraného času(3 sekundy) pošle openvpn servera čo i len jeden echo-response paket späť, znamená to, že sa podarilo openvpn spojenie a program mdpripoj úspešne končí. V opačnom prípade to znamená, že zlyhalo pripojenie na internet alebo na server. V takomto prípade sa mdpripoj pokúsi ukončiť pppd a openvpn. Bude sa na to snažiť použiť volanie kill a ich pid, ktoré sa bude snažiť získať zo súborov, kde by mali byť uložené. Nakoniec pošle mdpripoj RESUME príkaz programu MDemon a skončí s chybou.

K programu mdpripoj existujú ešte 2 podporné programy slúžiace na ukončovanie pppd a openvpn:

- mdclosepppd - pokúsi sa načítať pid z /var/run/ppp0.pid a poslať SIGTERM ppp daemonovi. V prípade úspechu sa pppd skončí, v prípade

[2] V prípade, že by sme sa pripojili do internetu cez sieťový interface pppn, číslo procesu, ktorý treba ukončiť na prerušenie spojenia, je uložené v /var/run/pppn.pid (n sa nahradí prirodzeným číslom). Systémy sa ohľadom implementácie pppd môžu od seba navzájom líšiť. Ubuntu 9.10, ktorí momentálne používam to rieši takto.

neúspechu je už skončený. Neskončiť sa nemusí v prípade ak closepppd nemal oprávnenie ukončiť pppd. Aplikácia sa snaží pred skončením vymazať /var/run/ppp0.pid. O vymazanie sa nepokúsi v prípade, že namala oprávnenie ukončiť pppd a pppd stále beží.

- mdcloseopenvpn - pokúsi sa z /usr/local/etc/openvpn.pid prečítať pid bežiaceho openvpn poslať mu SIGTERM. Ak sa ukončenie podarilo, mdcloseopenvpn vymaže /usr/local/etc/openvpn.pid. Ak sa nepodarilo, lebo openvpn už bol ukončený, vymaže súbor openvpn.pid. Súbor openvpn.pid ostane len v prípade, že mdcloseopenvpn sa nepodarilo vymazať kvôli právam, alebo mdcloseopenvpn nemalo oprávnenenie ukončiť openvpn.

5.7 mdpinger

Úlohou programu mdpinger je detekovať výpadok siete internet a v prípade detekcie výpadku, poslať sms administrátorovi pomocou programu MDemon.

Program mdpinger po spustení a daemonizáci uloží svoj pid do /usr/local/etc/mdemon/mdpinger.pid a spustí nekonečný cyklus. V tomto cykle každých 10 minút pošle 5 echo-request paketov na ip adresu googl.com a 5 echo-request paketov na ip adresu martinus.sk. Ak neobdrží do určitého časového limitu z daných adries ani jeden echo-response paket, pripojí sa mdpinger k programu mdemon a pošle sms administrátorovi. V prípade úspešného odoslania sms, mdpinger skončí. Ak sa nepodarí odoslať sms, mdpinger bude pokračovať vo svojom nekonečnom cykle.

Záver

V tejto práci som sa pokúsila navrhnúť a implementovať SMS bránu spolu s núdzovým systémom vzdialeného prístupu prostredníctvom mobilnej siete GSM/UMTS pod OS Linux. Vytvorila som tu jeden hlavný program MDemon s knižnicou pre jeho ovládanie, ktorá spoločne s programom MDemon umožňuje aplikáciam odosielanie SMS správ prostredníctvom pripojeného mobilného telefónu a vytvorenie vzdialeného prístupu pre administrátora využívajúceho na svoje pripojenie pripojený mobilný telefón. Okrem toho som vytvorila aj aplikácie umožňujúce ovládanie programu MDemon aj z iných k počítaču pripojených terminálov a aplikácie využívajúce program MDemon na umožnenie vzdialeného prihlásenia sa do systému v prípade nedostupnosti primárnych sieťových kanálov.

Príloha A

Súčasťou práce je CD, ktoré obsahuje zdrojové súbory programu MDemon, nástrojov pre ovládanie programu MDemon z konzoly, aplikácií využívajúcich program MDemon a knižnice pre aplikácie pracujúce s programom MDemon. Súčasťou CD je aj text tejto práce uložený v bakalarka.pdf v .pdf formáte.
Program MDemon spolu so zdrojovými súbormi a súborom Makefile pre kompiláciu sa náchádza v adresári mdemon. Programy predstavujúce nástroje pre ovládanie programu MDemon z konzoly sa nachádzajú v adresári mdemontools spolu so súborom Makefile pre ich kompiláciu.
Program MDemon spolu s nástrojmi na jeho ovládanie z konzoly sa dá zkompilovať a nainštalovať pomocou za sebou nasledujúcich príkazov:

```
./configure
make
make install
```

Aplikácie mdpinger a mdpripoj sa nachádzajú v adresári mdemonapplications spolu so súborom Makefile pre ich kompiláciu.

Literatúra

[1] Albert S. Huang, Larry Rudolph *Bluetooth Essentials for Programmers*, Cambridge University Press, 1 edition (September 3, 2007)

[2] http://en.wikipedia.org/wiki/Universal_Serial_Bus

[3] http://sk.wikipedia.org/wiki/Univerzálna_sériová_zbernica

[4] http://en.wikipedia.org/wiki/Hayes_command_set

[5] 3GPP TS 27.007 version 8.9.0 Release 8

[6] ETSI TS 100 585 V7.0.1 (1999-07)

[7] http://mobiletidings.com/2009/02/11/more-on-the-sms-pdu/

[8] Neil Matthew, Richard Stones *Beginning Linux Programming*, Wrox, 4th Edition (November 5, 2007)

[9] http://www.stunnel.org/

[10] http://stargate.cnl.tuke.sk/(tilda)mirek/homepage/pripojte-sa-do-internetu-z-mobilnej-siete-o2-cez-linux

[11] http://docs.telante.com/index.php/Advanced_GPRS_Modem_Usage

[12] http://docs.telante.com/index.php/APN

[13] http://en.wikipedia.org/wiki/Secure_Shell

[14] https://help.ubuntu.com/9.10/serverguide/C/openssh-server.html

[15] http://openvpn.net/index.php/open-source/documentation/