Robert Čordaš

CallTheTux

CallTheTux je univerzalni GSM stek za Linux. Dosadašnje implementacije namenjene su tačno jednom ili određenoj klasi modema. Univerzalnost se postiže pomoću lakog i efikasnog skript jezika kojim se opisuje rad modema. Podrška za nove telefone u dosadašnjim GSM engineima zahtevala je modifikaciju velikog dela izvornog koda, koji je teško razumljiv zbog načina na koji se odvija komunikacija sa modemom (npr. Openmoko-dialer (gsmd), Gommunicator, GPEPhone phone server). Ovaj projekat pojednostavljuje dizajniranje softverskog okruženja za mobilne telefone koji se baziraju na Linuxu i portovanje Linuxa na Windows i Symbian telefone. Sistem se bazira na klient-server arhitekturi. Za korisnički interfejs su zaduženi klijenti, što omogućuje da engine sistema radi pod bilo kojim grafičkom okruženjem (GPE, OPIE, stb).

Uvod

GSM modem je deo hardvera sa sopstvenim mikrokontrolerom koji komunicira sa glavnim procesorom telefona i zadužen je za bežičnu komunikaciju. Komunikacioni standard je određen AT Command Setom. U tom standardu je tačno opisan format komande, mogući odgovori modema na svaku komandu i njihovo značenje. Komande i odgovori koje modem daje su u ASCII formatu. Modem može da radi u dva moda: *data* modu i *command* modu. Kada je u *command* modu svaki karakter koji stigne do modema se interpretira kao deo komande. Posle *line termination* karaktera komanda počinje sa izvršavanjem. U data modu modem se nalazi nakon uspostavljanja konekcije. Tada se svaki podatak koji stigne do modema šalje putem uspostavljene konekcije.

AT Command Set je nastao kao skup komandi za prvi smartmodem (Hayes Smartmodem) i ubrzo je postao standard za komunikaciju sa smartmodemima. Vremenom se javila potreba za novim funkcijama i kompanije su dodavale sopstvene komande. Nastale su nove verzije standarda u kojima su neke komande izmenjene. Skoro svaka kompanija ima svoje specifične komande (Nokijin AT*NEONS koji služi za čitanje imena operatora iz SIM kartice ili AT*NCPS koji služi za restrikcije poziva, podešavanje identifikacije, itd.). Pored toga nijedan modem ne podržava kompletan AT Command Set.

Cilj ovog rada jeste da se definiše skript jezik u kome bi se opisivao rad konkretnog modema. Ideja je da GSM engine bude interpreter tog jezika i da se izvršavanjem skripte kontroliše modem. Mehanizam obrade komandi i odgovora modema je implementiran u interpreteru, dok se skriptom opisuju samo komande i odgovori. Tako se postiže jednostavno pisanje i modifikacija skripta, što pojednostavljuje rad programera. Prednost ovakvog pristupa je njegova univerzalnost. Time izbegavamo korišćenje raznih, uglavnom nekompletnih, GSM enginea (dobar primer je HTC Magician telefon gde Openmoko-dialer omogućuje primanje poziva, dok Gommunicator omogućuje slanje poruka i pozivanje, ali ne mogu raditi u isto vreme).

Dizajn i implementacija

Dizajnirani skript jezik mora da omogući jednostavan opis AT komandi. One moraju biti čitljive, a ipak brzo obradive. Interna obrada odgovora modema je vrlo složen proces. Odgovori mogu da sadrže razne tekstualne i numeričke informacije koje mogu biti potrebne za dalji rad modema (npr. status kod i slično).

Robert Čordaš (1989), Mužlja, Doža Đerđa 25, učenik 3. Razreda Zrenjaninske gimnazije

MENTOR: Dragan Toroman, ISP

Veliki problem je i to što je komunikacija sa modemom asinhrona. Može se dogoditi da na poslanu komandu stigne odgovor različit od očekivanog, jer se u bliskom vremenskom intervalu može desiti novi događaj (na primer: stigne ulazni poziv za vreme slanja sms poruke).

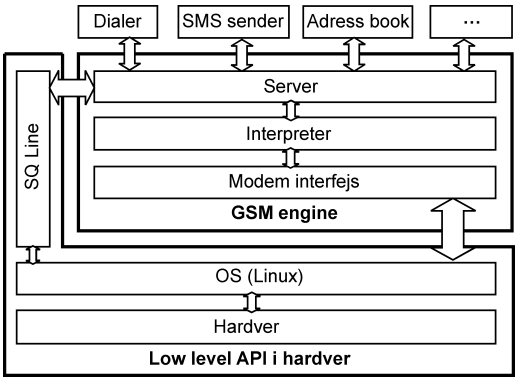
Potrebno je voditi računa i o potrebnim resursima (*CPU*, memorija), jer u najvećem broju slučajeva sistem radi na slabim ARM procesorima, čije performanse zaostaju za današnjim kućnim računarima.

Bitno je ostvariti modularnost sistema zbog njegovog lakog proširenja (lakog dodavanja novih funkcija). Stoga je sistem dizajniran po slojevima na koje se konektuju eksterni moduli.

Sistem se bazira na kilent-server arhitekturi. GSM engine je server, a klijenti su programi odgovorni za razne funkcije, uglavnom vezane za korisnički interfejs (npr. program za pozivanje brojeva (tzv. dialer), program zadužen za indikaciju dolaznog poziva ili poruke i sl.)

GSM engine se sastoji od 3 dela (slika 1):

1 Server je odgovoran za obradu ulaznih podataka i na osnovu njih šalje odgovarajuće komande interpreteru.
2 Interpreter izvršava skript kojim je modem opisan i prima komande koje dolaze od servera.
3 Interfejs modema je zadužen za komunikaciju sa modemom na najnižem nivou.



Slika 1. Struktura CallTheTux-a

Figure 1. CallTheTux structure

Skript jezik za opisivanje modema

Skripta za konkretan modem se prilagođava unapred zadatoj standardnoj formi – šablonu. Zadati su delovi skripta, imena, argumenti funkcija koje skript treba da sadrži da bi obavio neki zadatak. Sintaksa skript jezika podseća na pojednostavljenu sintaksu C jezika. Osnovne kontrolne strukture (*if, for* i *while*) imaju istu sintaksu kao u C-u. Varijable počinju karakterom $.

Funkcije se definišu na sledeći način:

```
ime_funckije($ime_argumenta1,
             $ime_argumenta2,....).
```

Stringovi mogu imati podstringove koji će za vreme izvršavanja skripte biti zamenjeni vrednošću neke zadate varijable (imena varijabli su u tom slučaju zadata između znakova #). Time je omogućena laka manipulacija stringovima.

U skriptu se u bilo kom trenutku mogu izvršiti komande nad modemom koristeći ključnu reč "command". U svakoj komandi je moguće koristiti varijable i svaka komanda ima sopstveni opis odgovora.

Opis skupa mogućih odgovora na jednu komandu su blokovi koji sadrže definicije odgovora (paterni). Oni mogu imati promenljive delove, koje interpreter može da predaje *handler* funkciji kao argument. Format patterna je:

```
fixni deo[tip:ime varijable]fixni
                                deo 2...
```

Na primer, za odgovor "test45abbilostakraj" i patern "test[int:broj]ab[string]kraj", varijabla po imenu "broj" bi dobila vrednost 45. Ako za neki deo definicije nije zadato ime varijable, ta informacija neće biti sačuvana. Tako se zanemaruju delovi odgovora koji nisu od interesa za izvršavanje skripta.

Struktura skripta se sastoji od 4 logičke celine:

1. Svaki skript sadrži globalnu tabelu događaja koja opisuje odgovore koje modem generiše na razne događaje. To su na primer odgovori koji signaliziraju ulazni poziv ili poruku. U toj tabeli se nalaze i delovi skripta koji se izvršavaju u slučaju da zadati pattern odgovara odgovoru. Izvršni delovi se koriste za signalizaciju događaja serveru. Na primer:

```
events{
  "+CREG: [int:stat]"{
    case_report $stat{
      0: OPERATOR_DISCONNECTED;
      1: OPERATOR_HOME_REGISTERED;
      2: OPERATOR_SEARCHING;
      3: OPERATOR_ACCESS_DENIED;
```

```
        5: OPERATOR_ROAMING;
      }
    }
};
```

2. Skript može da sadrži i takozvanu globalnu tabelu odgovora. Tu su opisani paterni kojima modem odgovara na komande. Takav je, na primer, odgovor "OK" za uspešno izvršenje komande. Tako se izbegava da isti odgovori budu više puta definisani.

```
global_responses{
  "OK"{
    report OK;
  }
};
```

3. Unapred određene funkcije koje izvršavaju specifične zadatke. Takve su na primer funkcije *dial(), enable(), disable()*, itd. Server automatski detektuje prisustvo određenih grupa funkcija i po tome određuje sposobnost modema. Na primer, ukoliko se detektuju funkcije koje se odnose na GPRS komunikaciju, GPRS funkcija će biti automatski dostupna.

```
dial($num){
  command "ATD#num#;"{
    responses {
      "NO CARRIER"{
         report NO_CARRIER;
      }
      "NO ANSWER"{
         report NO_ANSWER;
      }
      "BUSY"{
         report BUSY;
      }
    }
  }
}
```

4. Pomoćne funkcije koje mogu koristiti funkcije koje izvršavaju specifične zadatke

```
report_connection_status(){
  command "AT+CREG?"{
    "+CREG: [int:stat]"{
      case_report $stat{
        0: OPERATOR_DISCONNECTED;
        1: OPERATOR_HOME_REGISTERED;
        2: OPERATOR_SEARCHING;
        3: OPERATOR_ACCESS_DENIED;
        5: OPERATOR_ROAMING;
      }
    }
  }
};
```

Klijent

Klijenti su programi zaduženi za korisnički interfejs programa i razne dodatne funkcije (*auto-answer*, itd). Oni komuniciraju sa serverom pomoću protokola koji se bazira na TCP/IP protokolu. Zahtevi su formulisani na visokom nivou, nezavisnom od hardvera (modema) koji se koristi. Pre nego što klijenti postanu aktivni, registruju se na serveru kao programi za obavljanje nekih funkcija. Kasnije server zna koje događaje kom klijentu treba da javi. Server koristi klijente za svaku interakciju sa korisnikom. Tako je korisnički interfejs potpuno odvojen od servera, čime je omogućeno da sistem radi na bilo kom grafičkom okruženju.

Zbog korišćenja TCP/IP protokola su neophodne neke sigurnosne mere, pošto se serveru može pristupiti sa udaljenih mašina, što predstavlja problem kod sistema sa Internet konekcijom (GPRS).

Interpreter

Interpreter je zadužen za izvršavanje skripta koji opisuje modem. Skriptni jezik se interpretira zbog njegovog visokog nivoa. Kompajliranje bi bilo mnogo složenije, a u praksi performanse izvršavanja se ne bi poboljšale. Razlog za to je što u većini slučajeva blokovi koji se izvršavaju kao reakcija na događaj ili komandu sadrže svega nekoliko instrukcija.

Prilikom inicijalizacije, parser generiše stablo zavisnosti od izvornog koda skripte. Svakom čvoru se dodeljuje *handler* funkcija koja ga obrađuje. Dodela se radi na osnovu definicija koje su paterni za delove stabla. Parser izvršava top-down pattern matching na stablu. Tako svaki čvor dobija svoju handler funkciju, funkciju za inicijalizaciju i funkciju za dealokaciju. Kada je pattern matching završen, parser poziva sve funkcije za inicijalizaciju u bottom-up redosledu. Time se postiže da su u trenutku inicijalizacije nekog čvora svi njegovi podčvorovi već inicijalizovani. Funkcije za inicijalizaciju pretvaraju potrebne argumente u formu pomoću koje se odgovori mnogo efikasnije mogu obraditi za vreme izvršavanja: stringovi koji sadrže promenljive delove se konvertuju u liste fiksnih delova i reference na varijable. Odgovori se preprocesiraju na način opisan u poglavlju Pattern matching odovora.

Za interpretiranje koda se kreira novi thread interpretera. Moguće je instancirati više interpretera u isto vreme od kojih je samo jedan aktivan dok su ostali suspendovani.

Pattern matching odovora

Problem pattern matchinga odgovora modema je nedeterminsitički problem, pošto se u toku sekvencijalne obrade karaktera odgovora ne može tačno odrediti kraj promenljivog dela. Na primer ako interpreter obrađuje odgovor "testbcdabcabc" po definiciji "test[string]abc", intepreter će na poziciji posle "testbcd" (pogrešno) detektovati kraj prvog tekstualnog argumenta. Razlog za to je što se zadnji deo argumenta poklapa sa fiksnim delom patterna odmah posle argumenta. Taj problem je rešen korišćenjem backtrackinga. Stanje pattern matchinga se zapamti posle kraja svakog argumenta. U slučaju da se neki deo patterna ne može matchovati, matcher se vraća na poslednji element i proba da obradi sledeći karakter kao deo argumenta, a ne kao deo fiksnog dela patterna. Višelinijski patterni dodatno komplikuju obradu.

U praksi se mogući odgovori od modema skoro nikada ne preklapaju ili postoji minimalno preklapanje. Zbog toga se pronalaženje odgovarajućeg patterna može rešiti sekvencijalnom pretragom tabele mogućih odgovora. Ako pattern matcher detektuje karakter u stringu koji se ne može matchovati, odmah se prolazi na sledeći potencijalni pattern.

Obrađivanje odgovora

Pošto je komunikacija sa modemom asinhrona, mora se utvrditi da li je dobijeni string od modema odgovor na poslatu komandu ili izveštaj o nekom događaju koji se dogodio u međuvremenu. To se utvrđuje na osnovu baze (baze odgovora ili baze događaja) u kojoj je nađen pattern koji odgovara odgovoru. Ako je izveštaj o događaju, posle njega u baferu modema mora da sledi odgovor na izvršenu komandu. Ukoliko je komanda bila zadnja izvršena akcija na modemu, prvo se njegova obrada mora završiti. Interna logika servera odlučuje da li se komanda nastavlja, ili se njeno izvšavanje ukida (na primer ako je stigao RING signal od modema (koji signalizira ulazni poziv, a poslata je ATD komanda za pozivanje broja, nema smisla da se nastavlja poziv). Kada je odgovor na komandu obrađen, intepreter koji je izvršio komandu se suspenduje (ako se komanda nastavlja), i pokreće se novi interpreter koji obrađuje odgovor modema. Kad se obrada završi, sa radom nastavlja interpreter-thread koji je pozvao komandu.

Kada je pronađen pattern koji odgovara odgovoru modema, njegov handler se interpretira. Glavni zadatak skripta je da konvertuje odgovor u format koji server može razumeti (ako je potrebno vodeći računa i o prethodnim odgovorima) i da ih prosledi serveru.

Interfejs modema

Ovaj modul obezbeđuje komunikacioni API sa modemom na niskom nivou. Pomoću njega se GSM engine može potpuno odvojiti od ulazno izlaznih funkcija specifičnih za određeni tip modema. Zadatak mu je samo da prosledi modemu komande koji stižu od interpretera i da prosledi odgovore modema interpreteru. To je potrebno pošto se modemu pristupa na različite načine zavisno od drajvera modema. U većini slučajeva ovaj intefejs je samo wrapper za I/O funkcije modema, ali na primer kod HTC Magician telefona prvo se mora selektovati pernosni kanal.

Server

Server obezbeđuje vezu između interpretera i različitih klijenata, i unutrašnju logičku kontrolu. Glavni zadaci su mu:

- Obrađivanje zahteva klijenata
- Praćenje i obrada odgovora koji stižu od modema
- Ostali zadaci: upravljanje listom poziva, čuvanje ulaznih i izlaznih poruka, itd.

Ako server dobije zahtev od klijenta onda ih on obrađuje pomoću interpretera.

Server kontinualno čeka za odgovore od strane modema. Kada detektuje neki događaj, izvršava pattern matching algoritam (opisano u sekciji Pattern matching odovora), i pokreće intepreter thread koji ga obrađuje.

Kada do servera stigne posleđeni odgovor obrađen sa strane interpretera, njegova interna logika određuje reakciju. Postoje događaji koji se mogu obraditi interno (ako se događaj odnosi na neku servis funkciju, na primer potreba za ponovnom registracijom na mreži), ili može da traži pomoć klijenta (ako je to neki globalni događaj ili ako zahteva interakciju sa korisnikom).

Server je zadužen i za čuvanje ulaznih i izlaznih poruka i log fajl poziva. To omogućava da svaki klijent može pristupati tim podacima. Podaci se čuvaju u SQLite bazi.

Zaključak

Problem GSM steka je jedan od najvećih problema koji se javljaju pri korišćenju Linuxa na mobilnim telefonima. Pomoću CallTheTuxa je omogućeno lakše portovanje okruženja za Linux operativni sistem na mobilne telefone. Izbegava se modifikovanje već postojećih, uglavnom komplikovanih sistema. Realizovan je jedan modularan sistem koji je univerzalan za sve telefone i čija se funkcionalnost lako može proširivati. Modifikacijom klijenata, CallTheTux se može koristiti na raznim grafičkim okruženjem (GPE, OPIE, OPIE II, itd.). Rad modema se opisuje veoma jednostavnim skript jezikom koji podržava AT Command Set. Engine je dovoljno mali i brz da bi se mogao koristiti na bilo kom procesoru koji se danas koristi u mobilnim telefonima.

Literatura

Mitchell M., Oldham J., Samuel A. 2001. *Advanced Linux Programming*. USA: New Riders Publishing

http://en.wikipedia.org/wiki/Hayes_command_set

Standardi:

AT Command Set for Nokia GSM products: http://www.itruss.com/files/atnokia.pdf

AT Command Reference Manual: http: //www.data-com-experts.com/AT_commands.pdf

3GPPAT command set for User Equipment: http://www.3gpp.org/ftp/Specs/archive/ 27_series/27.007/27007-3d0.zip

Tutoriali:

Scalable Network Programming: http://bulk.fefe.de/scalable-networking.pdf

POSIX threads programming tutorial: http://www.llnl.gov/computing/tutorials/pthreads/

Linux Headquarters: Introduction to GTK+ Programming: http://www.linuxheadquarters.com/howto/programmin g/gtk_examples/index.shtml

Robert Čordaš

CallTheTux

CallTheTux is an universal GSM stack for Linux. GSM engines used nowdays are intended only for one type of GSM modem or a small subclass of similar modems. Universality is achieved with an easy and efficient script language which describes the operation of the modem. Adding support for a new GSM modem in the present GSM engines requires a lot of modifications in the code. The source code itself is hard to understand because of the way of the communication with the modem. This project simplifies the design of software environments for mobile phones based on Linux and porting the Linux to Windows and Symbian based phones. The engine is based on a client-server architecture. Clients are responsible for the user interface. This enables the engine to work under all graphical environments without modification.

The whole communication, command and response parsing and conflict-avoiding mechanism is built into the engine, the script only describes the commands and the possible responses with predefined patters automatically, so the needed status codes can be extracted very easily.

The problem of GSM stack is one of the most important problems in using Linux on cellphones. With CallTheTux the porting of the Linux system to mobile phones becomes much easier. It makes the modification of already existing, very complicated and incomplete systems, unnecessary. The modular system created is universal for all mobile phones. The functionality of the system can be easily extended. It can be used in various desktop environments (GPE, OPIE, OPIE II, OpenMoko, etc.) just by modifying the clients. The modems are described with a very simple script language which is capable of describing any command and response from the AT Command Set. The engine is small and fast, so it can be used on all processors used in mobile phones nowadays.

Prilog: gramatika jezika

```
identifier = [a-zA-Z][a-zA-Z0-9_]*
number = [0-9]+ | 0[xX][0-9A-Fa-f]+
string = "[\x20-\x7E]"
variable = $[a-zA-Z][a-zA-Z0-9_]*

operator = '+' | '-' | '*' | '/' | '%' | '|' | '&' | '>' | '<' | '' | '' |
'==' | '=' | '' | '!=' | '='
unary_operator = '-' | '~'
postfix_operator = '++' | '--'

keyword = 'command' string
             | 'report' status

expr = variable
             | number
             | '(' expr ')'
             | expr operator expr
             | unary_operator expr
             | variable postfix_operator
             | function_call

stmt = 'if' '(' expr ')' block
             | 'for' '(' expr? ';' expr? ';' expr?) block
             | 'while' '(' expr ')' block
             | 'do' stmts 'while' '(' expr ')'
             | function_call
             | variable '=' expr
             | 'switch' '(' expr ')' '{' (expr ':' block)+ '}'
             | keqword
             | block

stmts = stmt
             | stmts ';' stmt

block = stmt
             | '{' stmts '}'

function_call = identifier '(' (expr | ((expr ',')+ expr)) ')'
function_def = identifier '(' (variable | ((variable ',')+ variable)) ')' '{'
stmts '}'

response_defs = (string '{' stmts '}')+

global_events = 'events' '{' response_def '}' ';'
global_responses = 'global_responses' '{' response_def '}' ';'

program = global_events
             global_responses
                                fucntion_def+
```