SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 18

SUSTAV ZA ODGOĐENO POKRETANJE POSLOVA

Krunoslav Tomorad

Zagreb, lipanj 2010.

Sadržaj

1. Uvod

Cilj je ovog rada dati teorijsku podlogu te napraviti konkretno ostvarenje programa za pokretanje drugih programa s vremenskom odgodom. Osnova ovakvih programa su tzv. poslovi (engl. *jobs*) koji se sastoje od vremenske oznake te programa koji se pokreće. Tipične su im mogućnosti periodičko pokretanje tih poslova; pri tome interval pokretanja može biti jednostavan (primjerice jednom na mjesec) ili složen (npr. u osam sati ujutro svakoga radnog dana). Primjeri korištenja uključuju periodičko provjeravanje elektroničke pošte, automatsko ažuriranje programa, obavještavanje o nadolazećim događajima itd.

Iako postoji više programa te namjene za GNU/Linux operacijski sustav, nijedan od njih ne pruža programsko sučelje (API, engl. *application programming interface*) što otežava rukovanje poslovima iz drugih programa. Također, većina njih pretpostavlja kako je računalo stalno uključeno iako je vjerojatnije da na osobnim računalima to nije tako. Ti nedostatci bili su motivacija za izradbu novog ostvarenja.

Nakon kratkog pregleda trenutnih ostvarenja u drugom poglavlju, u trećem su poglavlju opisani GNU razvojni sustav i programske knjižnice. Detalji ostvarenja nalaze se u četvrtom poglavlju. U dodatku je na jednom jednostavnom primjeru pokazan postupak korištenja GNU razvojnog alata.

2. Razvoj ostvarenja kroz povijest

U vrijeme kada se pojavio prvi poznati program ove namjene (`cron`, krajem '70-ih), računala nisu bila česta te su se većinom nalazila na sveučilištima i u velikim tvrtkama. Bila su to relativno velika računala koja su se u načelu gasila samo iznimno. Kada se razmatraju programi ove namjene, može se vidjeti kako su ovo gotovo idealni uvjeti za njih. Zato ne čudi što je `cron` bio prilično jednostavan i malen. Kako je računalna snaga rasla, a računala su postala dovoljno mala i pristupačna za široku upotrebu, pokazalo se kako `cron` i njemu slična ostvarenja više nisu zadovoljavajuća. Mogućnosti izvornog `crona` su proširene, a sukladnost s njim je više-manje očuvana.

Danas svaki operacijski sustav ima barem jedno ostvarenje programa za vremenski uvjetovano pokretanje drugih programa. Ostvarenja se međusobno razlikuju po izvedbi, mogućnosti, sučelju, prenosivosti i po drugim svojstvima. Cilj je ovog poglavlja ukratko opisati neka od najkorištenijih ostvarenja.

2.1. Cron

Program `cron` je izvorno ostvarenje programa ove namjene. Prva je inačica `crona`, koja se pojavila u Unixu inačice 7 (engl. *Version 7 Unix*), dopuštala njegovo korištenje samo korisniku *root* – to je bila jednokorisnička inačica. Osnova je programa konfiguracijska datoteka (tablica) naziva `crontab`. Datoteka se sastoji od redaka koji predstavljaju poslove (engl. *cron jobs*), a svaki se posao sastoji od dva osnovna dijela: vremenska oznaka i program koji će se izvršiti. Vremenska je oznaka izraz s kojim će `cron` svake minute pokušati podudariti trenutni datum i vrijeme – ako se ta dva vremena podudaraju, program će se izvršiti, u suprotnom neće. U dijelu s programom nalazi se putanja do programa, koji će se izvršiti, zajedno s parametrima koji se prilikom izvršavanja prosljeđuju tome programu.

Izvorni je `cron` radio po sljedećem algoritmu:

1. pročitaj datoteku `/usr/etc/crontab`;
2. odredi podudara li se vremenska oznaka kojeg posla s trenutnim vremenom te u ovisnosti o tome izvrši odgovarajuću naredbu kao korisnik *root*;
3. odspavaj jednu minutu;
4. vrati se na prvi korak.

Glavni je nedostatak ovog algoritma bila njegova jednostavnost – svake minute se troši procesorsko vrijeme bez obzira na to podudara li se koja vremenska oznaka ili ne. S obzirom na to da su, u vremena kada se ta inačica `crona` koristila, računala imala male procesne mogućnosti, ta je činjenica predstavljala popriličan problem. U to je vrijeme (kasne '70-e) bio napravljen pokus na Sveučilištu u Purdueu u kojem je dostupnost `crona` bila proširena na svih 100 korisnika te je tim pokusom pokazano kako `cron` koristi previše procesorskog vremena.

Prekretnicu u širokoj uporabi `crona` izazvao je članak iz kolovoza 1977. godine objavljen u časopisu *Communications of the ACM* (Franta, 1977.). U tom je članku opisana lista događaja, struktura podataka namijenjena diskretnim, događajima uvjetovanim, simulacijskim sustavima. U usporedbi s povezanom listom, lista događaja ima manju složenost dodavanja. Dok je složenost te operacije kod povezane liste $O(n)$, kod liste događaja u najgorem je slučaju $O(\sqrt{n})$. Ubrzo se pojavila inačica `crona` koja je koristila tu novu strukturu podataka i time uvelike poboljšala učinkovitost izvorne inačice `crona`, a usto je dopuštala svim korisnicima njezino korištenje. Tu su inačicu napisali Robert Brown i Keith Williamson na Sveučilištu u Purdueu 1979. godine.

Algoritam poboljšane inačice `crona` je sljedeći:

1. prilikom pokretanja, pretraži sve datoteke imena `.crontab` u početnim kazalima korisnikâ;
2. za svaku pronađenu datoteku i za svaki posao u njoj odredi sljedeće vrijeme u budućnosti kad se posao treba izvršiti;
3. dodaj te poslove, zajedno s pripadajućim vremenskim oznakama i korisničkim imenima te vremenima sljedećeg izvršavanja u Franta-Malyjevu listu događaja;
4. uđi u glavnu petlju:
    1. pogledaj posao na početku liste te odredi za koliko se vremena treba izvršiti;
    2. odspavaj tu količinu vremena;
    3. nakon buđenja, izvrši u pozadini program s početka liste s povlasticama korisnika kojem taj posao pripada;
    4. ažuriraj vrijeme sljedećeg izvršavanja tog posla te ga, tako izmijenjenog, premjesti na odgovarajuće mjesto u listi događaja.

Program `cron` je dalje bio poboljšan u Bellovim laboratorijima gdje su konfiguracijske datoteke iz početnih kazala korisnika prebačene u zajedničko kazalo. Ta je inačica dugo vremena bila najzastupljenija na Unix operacijskim sustavima; danas se većinom koristi inačica Paula Vixieja iz 1993.

2.2. At

Program `at` još je jedan program s Unix operacijskog sustava čija je glavna razlika u odnosu na `cron` što se zadani posao izvršava samo jednom, a ne periodički. Također, `at` čuva okolinu u kojoj je posao dodan, primjerice radno kazalo i varijable okoline. Program `at` na ulazu prima niz naredbi i vremensku oznaku te takav posao stavlja na čekanje sve dok se trenutno vrijeme ne poklopi s vremenskom oznakom. U trenutku kada posao treba izvršiti, `at` će izvršiti sve naredbe iz niza naredbi, u očuvanoj okolini.

Windows operacijski sustavi također imaju naredbu `at` (`at.exe`) no njena je funkcionalnost sličnija programu `cron` nego `at`. Također, Windows Task Scheduler je noviji i napredniji program te namjene za Windows operacijske sustave.

2.3. Anacron

Iako je `cron` načelno dobro razrađen program, on ima i svoje nedostatke. Jedan od najvažnijih je pretpostavka kako je računalo stalno uključeno. Program `anacron` ispravlja taj nedostatak. Pritom je važno napomenuti kako `anacron` nije samostalni program, već on ovisi o `cronu`.

Taj nedostatak dolazi do izražaja kod korištenja poslova koji su inače namijenjeni poslužiteljima. Naime, poslužitelji su u načelu najmanje opterećeni preko noći pa su i mnogi rutinski poslovi postavljeni na način da se pokreću upravo u to doba dana. S druge strane, osobna su računala većinom uključena danju, stoga nije nemoguć scenarij u kojem se neki posao nikad ne izvede na osobnom računalu.

Kako bi riješio taj problem, `anacron` dopušta vremensku marginu kod pokretanja posla. Ako se neki posao nije izvršio jer računalo nije bilo pokrenuto, izvršit će se ubrzo nakon pokretanja računala.

Nedostatak `anacrona` je što samo korisnik *root* može dodavati `anacron` poslove, za razliku od `crona` kod kojeg bilo koji korisnik može dodavati poslove. Usto, dok `cron`

ima razlučivost vremenske oznake od jedne minute (znači, trenutak izvršavanja može se odrediti s preciznošću od jedne minute), razlučivost `anacrona` je tek jedan dan.

2.4. Launchd

Program `launchd` sastavni je dio Mac OS X operacijskog sustava koji ujedinjuje više sustavskih alata, uključujući i `cron`. Iako je dio `launchda`, koji je zadužen za vremenski uvjetovano pokretanje programa, funkcionalnošću gotovo identičan `cronu`, način korištenja mu je u potpunosti drugačiji. Kao i svi drugi dijelovi `launchda`, i poslovi `crona` predstavljeni su takozvanom *p*-listom koja je u biti XML datoteka.

Dobar strana ujedinjavanja raznih alata u jedan je gotovo identičan način podešavanja svih tih programa. To rezultira dosljednošću i jednostavnošću podešavanja sustava. Prednost `launchda` u odnosu na `cron` je i što ima programsko sučelje, pa je dodavanje novih poslova iz drugih programa izrazito jednostavno i svodi se na pozivanje odgovarajuće funkcije.

2.5. Windows task scheduler

Ranije je spomenut alat `at.exe` na Windows operacijskim sustavima. Task Scheduler modernije je rješenje koje se koristi na tim operacijskim sustavima. Task Scheduler je istovjetan `cronu` na Unixu sličnim operacijskim sustavima. Prvi se put pojavio kao dodatak za Windowse 95, a postao je sastavnim dijelom Windowsa u inačici 2000. Riječ je o grafičkom sučelju te se dodavanje poslova i sve podešavanje odvija preko grafičkog korisničkog sučelja. Sami poslovi se spremaju u binarnu datoteku s proširenjem `.job` (novije inačice koriste XML). Trenutna je inačica Task Scheduler 2.0 i uvedena je u operacijskom sustavu Windows Vista.

3. Potrebni alati i postupci

Kod izradbe inačice `crona` potrebno je znati neke postupke, strukture podataka kao i poznavati alate za izradbu aplikacija prenosivih na razne inačice *Unix* operacijskih sustava. To uključuje korištenje *GNU Autotools* alata te korištenje i izradbu knjižnica. Cilj je ovog poglavlja dati uvid u postupke koji su od važnosti za izradbu ovakvog programa.

3.1. GNU razvojni sustav

GNU razvojni sustav, poznat i kao *GNU Autotools*, skup je programskih alata razvijenih u okviru GNU projekta čiji je cilj pojednostavljenje postupka održavanja programskih projekata prenosivim na raznim Unixu sličnim operacijskim sustavima. Najbitniji alati ovog sustava su *Autoconf*, *Automake* i *Libtool*. Sustav je jako raširen – većina projekata slobodnog koda ga koristi. Krajnjem korisniku ovaj sustav daje mogućnost izgradnje i instalacije programskog projekta u tri koraka: 1) pokretanje `configure` skripte, 2) izvršavanje naredbe `make` te 3) pokretanje naredbe `make install`. Prvim se korakom projekt podešava za sustav na kojem se izvršava, a proizvod je tog podešavanja `Makefile` datoteka. U drugom koraku program `make` izvršava naredbe iz novostvorene `Makefile` datoteke. Na kraju, u trećem koraku, pokreće se *install* meta u `Makefile` datoteci te se time izgrađeni projekt instalira, tj. pripadajuće se datoteke kopiraju u standardna kazala.

Autoconf alat obrađuje datoteku `configure.ac` te iz nje stvara `configure` skriptu, koja ispituje platformu na kojoj se program instalira, s ciljem da korisnika oslobodi podešavanja parametara povezanih s prenosivošću. *Automake* stvara `Makefile` predloške. Na ulazu prima datoteku `Makefile.am`, a na izlazu daje datoteku `Makefile.in` koju `configure` skripta koristi kod stvaranja krajnje `Makefile` datoteke. *Libtool* postoji kako bi se olakšalo stvaranje programskih knjižnica na različitim platformama jer se način njihova stvaranja razlikuje među operacijskim sustavima.

S obzirom na to da korisnik i alati GNU razvojnog sustava stvaraju velik broj datoteka, bilo bi nepregledno stavljati sve datoteke u jedno kazalo. Zbog toga je običaj organizirati programski projekt slijedeći konvenciju GNU razvojnog sustava. Neka od često korištenih dodatnih kazala su:

- `doc/`: u ovo se kazalo sprema bilo kakva dokumentacija vezana za program (bilo dokumentacija koda ili dokumentacija za krajnjeg korisnika);
- `src/`: ovdje se stavlja programski kôd – ukoliko je program složen, preporuka je organizirati ga dalje po kazalima;
- `lib/`: ovdje se stavlja kôd vezan uz prenosivost, npr. zamjene za funkcije koje nisu dostupne na pojedinim platformama;
- `m4/`: ovdje se mogu staviti dodatne `m4` datoteke kojima se definiraju nove naredbe programa `autoconf`.

Također, GNU razvojni alat zahtijeva postojanje nekih datoteka. To su datoteke:

- `README`: tu se nalaze najvažnije informacije o programu te reference na ostalu dokumentaciju;
- `INSTALL`: s obzirom na to da je postupak izgradnje i instalacije programa koji koristi GNU razvojni sustav isti za sve takve programe, sustav stvara standardnu `INSTALL` datoteku, no ako se postupak instalacije razlikuje od standardne, tada se pripadajuće upute stavljaju u ovu datoteku;
- `AUTHORS`: ova bi datoteka trebala sadržavati popis svih osoba koje su pridonijele projektu (što je korisno zbog autorskih prava);
- `NEWS`: sadrži popis bitnih značajki koje su se pojavile u pojedinoj inačici programa;
- `ChangeLog`: koristi se za bilježenje svih promjena koda.

Primjer korištenja GNU razvojnog sustava nalazi se u dodatku.

3.2. Programske knjižnice

S porastom složenosti programa pojavila se potreba za odvajanjem zajedničkog koda u zasebne datoteke. Time se smanjuje uvišestručenje koda, a programu se dodaje mogućnost modularnosti. Te se datoteke nazivaju programskim knjižnicama (engl. *program library*). Knjižnice se sastoje od prevedenih funkcija i razreda koje drugi programi mogu koristiti. Primjer je standardna knjižnica programskog jezika C (`libc`).

Kad neki program koristi neku knjižnicu, on se mora s tom knjižnicom povezati (engl. *link*). Program se povezuje s knjižnicom prilikom prevođenja (engl. *compile time*) ili prilikom pokretanja (engl. *run time*). Povezivanje je obično odvojeno od prevođenja,

program koji to radi zove se povezivač (engl. *linker*). Nakon što je program preveden i povezan, može se pokretati. Ovisno o vrsti knjižnice (što je dolje objašnjeno), prilikom pokretanja programa može biti potreban dodatan korak, kod kojeg se knjižnica učitava u radnu memoriju te povezuje s programom, koji koristi funkcije iz te knjižnice. Taj dio obavlja program koji se naziva punilac (engl. *loader*).

Ovisno povezuje li se program s knjižnicom u trenutku prevođenja ili tijekom izvođenja programa, govorimo o statičkom i dinamičkom povezivanju knjižnice. Dinamičko povezivanje knjižnica se u načelu koristi kada program treba imati mogućnost dodataka (engl. *plug-in*).

Knjižnice moraju biti podržane na razini operacijskog sustava, a s obzirom na smještaj koda knjižnice u odnosu na program mogu se podijeliti na dvije osnovne vrste: dijeljene knjižnice i statičke knjižnice. Danas prevladavaju dijeljene knjižnice. Razlika među njima je i na izvedbenoj razini i u načinu rada. Statičke su knjižnice, zapravo, arhive u kojima se nalazi čisti prevedeni kôd funkcija (tzv. objektni kôd). Prilikom prevođenja i povezivanja nekog programa, objektni se kôd funkcija, koje se u tom programu koriste (a koje se nalaze u knjižnici), prepisuje u objektni kôd prevedenog programa. Nedostatak tog pristupa vidljiv je ukoliko više programa koristi funkcije iz iste knjižnice. Tada će se u svakog od njih prepisati isti vanjski objektni kôd, a nakon pokretanja više tih programa, podatkovni dijelovi pripadajućih procesa bit će nepotrebno povećani zajedničkim kodom, koji bi inače mogao biti na jednom mjestu. To je vrlo realna situacija; primjerice najveći dio programa pisanih u programskom jeziku C koristi njegove standardne funkcije koje se nalaze u jednoj knjižnici.

S druge strane, dijeljene knjižnice (engl. *shared object library*) nisu izvedene kao arhive, već koriste posebni oblik izvršne binarne datoteke. Dio s povezivanjem prevedenog programa s dijeljenom knjižnicom isti je kao i kod statičkih knjižnica. Razlika je kod prevođenja programa i kod njegovog učitavanja. Kod dijeljenih knjižnica objektni se kôd pripadnih funkcija ne kopira u objektni kôd programa, već se prilikom pokretanja programa prije njega u memoriju učita ta knjižnica. Pri tom se knjižnica ne učitava u podatkovni dio procesa nastalog iz pokrenutog programa, već u svoj vlastiti dio memorije. Ako se nakon toga pokrene neki drugi program koji koristi

istu knjižnicu, ona se neće ponovno učitati u memoriju, već će i taj program biti povezan s već učitanom knjižnicom.

Knjižnice se u programskom jeziku C stvaraju iz izvornog koda spremljenog u običnim `.c` datotekama, s razlikom što taj kôd nema funkciju `main()`. Iz tog je razloga potrebno obavijestiti programski prevodilac o tome, u suprotnome će zaustaviti prevođenje kad ustanovi da funkcija `main()` nedostaje. Također mu je potrebno prenijeti i druge parametre kako bi ispravno napravio kôd pogodan za stvaranje knjižnice.

U nastavku je opisan postupak stvaranja jednostavne dijeljene knjižnice te njezine uporabe na GNU/Linux operacijskom sustavu. Koristi se programski prevodilac `gcc`.

Sljedeći će se programski kôd napisan u jeziku C prevesti u dijeljenu knjižnicu:

```
$ cat knjiznica.c
#include <stdio.h>

int ispisi_pozdrav(char ime[])
{
    printf("Zdravo %s! Kako si?\n", ime);

    return 0;
}
```

Prvo što treba napraviti je prevesti izvorni kôd `knjiznica.c` u objektni kôd. To se radi na sljedeći način:

```
$ gcc -c -fPIC -o knjiznica.o knjiznica.c
```

Odrednica `-c` govori prevodiocu da ne pokreće program poveznik. Time se izbjegava traženje funkcija iz vanjskih knjižnica, koje program eventualno koristi, kao i traženje funkcije `main()`. Odrednicom `-fPIC` se prevodioca obavještava da stvori kôd neovisan o položaju u memoriji (engl. *position independent code*). To je nužan zahtjev koji nameću dijeljene knjižnice.

Nakon prevođenja u objektni kôd, pristupa se izradbi same knjižnice. Postupak je sljedeći:

```
$ gcc -shared -Wl,-soname,libpozdrav.so.1 -o libpozdrav.so.1.0.0
knjiznica.o
```

U prethodnoj naredbi, odrednica `-shared` je potrebna radi stvaranja dijeljene knjižnice, a zatim slijedi odrednica `-Wl`. Sve što se nalazi u nastavku te odrednice

prevodilac će proslijediti programu za povezivanje kao njegove odrednice. Zarezi pri tome služe kako bi se povezivaču moglo poslati više od jedne odrednice, u ovome slučaju, prenosi se odrednica `-soname` kojom se govori povezivaču da iza nje slijedi naziv knjižnice. Slijedi odrednica prevodioca `-o`, iza koje se nalazi fizičko ime datoteke s dijeljenom knjižnicom, a na kraju se nalazi popis objektnih datoteka od kojih se knjižnica sastavlja.

Iako bi se kao naziv knjižnice mogao staviti proizvoljan niz znakova, takva knjižnica ne bi slijedila nazivlje knjižnica u GNU/Linux operacijskom sustavu te bi ju bilo nemoguće koristiti. Nazivlje propisuje ime knjižnice na sljedeći način: prvo dolazi znakovni niz `lib`, iza čega slijedi proizvoljan naziv knjižnice, zatim proširenje `.so` (od engl. *shared object*) i točka iza njega te na kraju inačica knjižnice. Inačica se sastoji od tri broja odvojenih točkom. Prvi broj označava tzv. glavnu inačicu knjižnice (engl. *major*) i on se smije uvećati samo ako se programsko sučelje koje knjižnica pruža mijenja. Sljedeća dva broja predstavljaju sporednu inačicu programa (engl. *minor*) te broj izdanja (engl. *release number*). Kako bi program punilac mogao pronaći knjižnicu, potrebno je napraviti dvije simboličke poveznice koje pokazuju fizičku datoteku knjižnice:

```
$ ln -s libpozdrav.so.1.0.0 libpozdrav.so.1
$ ln -s libpozdrav.so.1 libpozdrav.so
```

Sama knjižnica nije dovoljna za njezino uspješno korištenje. Program, koji će koristiti funkcije iz te knjižnice, mora znati koji su im prototipovi. Zbog toga je potrebno odgovarajuće zaglavlje s prototipovima funkcija:

```
$ cat knjiznica.h
int ispisi_pozdrav(char []);
```

Nakon što je knjižnica prevedena i napisano je odgovarajuće zaglavlje, može se napisati program koji koristi tu knjižnicu. Program u nastavku jednostavno poziva funkciju `ispisi_pozdrav()` te joj kao parametar prenosi prvi argument naredbenog retka.

```
$ cat program.c
#include <stdio.h>
#include "knjiznica.h"

int main(int argc, char *argv[])
{
    ispisi_pozdrav(argv[1]);

    return 0;
}
```

Program se prevodi na sljedeći način:

```
$ gcc -o program program.c -L. -lpozdrav
```

Odrednicom `-L` prevodiocu se govori da knjižnice koje slijede traži i u trenutnom kazalu (to je određeno točkom iza odrednice), a ne samo u standardnim kazalima. Na kraju slijedi odrednica `-l`, iza koje se navodi ime same knjižnice koja se koristi, bez predmetka `lib`, proširenja i inačice. Pri tome, knjižnica koja se koristi može biti ili statička ili dijeljena, a ako postoji ista knjižnica u oba oblika, prevodilac će odabrati dijeljenu.

Program se sada može pokrenuti, no učitavač programa neće moći pronaći knjižnicu ako nije u nekom od standardnih kazala (npr. `/usr/lib/`), osim ako mu se eksplicitno drugačije ne kaže. Budući da se knjižnica iz primjera nalazi u kazalu s izvornim kodom, prilikom pokretanja programa potrebno je postaviti varijablu imena `LD_LIBRARY_PATH` s imenom kazala kao vrijednošću. Program se pokreće na sljedeći način:

```
$ LD_LIBRARY_PATH=. ./program Medo
Zdravo Medo! Kako si?
```

4. Arhitektura ostvarenja

U ovome poglavlju opisana je organizacija koda ostvarenja, objašnjen je oblik posla programa `cron` te su navedene smjernice kojima se vodilo prilikom oblikovanja ostvarenja.

Ostvarenje je napisano u programskom jeziku C za POSIX sustave, što uključuje i GNU/Linux operacijski sustav. Izvorni program ove namjene zove se *cron*, što dolazi od jedne grčke riječi za vrijeme. Stoga je kao naziv ostvarenja izrađenog u okviru ovog rada odabrana druga riječ, *keros* (izvorno *καιρός*).

4.1. Organizacija programskog koda

Ostvarenje je organizirano u tri cjeline: pozadinski proces (engl. *daemon*, naziva `kerosd`), knjižnica (`libkeros`) te alat naredbenog retka (`kerostab`). Sva bitna programska logika nalazi se u pozadinskom procesu koji je samostalan (tj. ne ovisi o preostala dva dijela). Knjižnica definira i ostvaruje programsko sučelje, koje korisnički programi mogu koristiti za dohvaćanje liste korisničkih poslova, za brisanje korisničkih poslova te za zamjenu trenutnih korisničkih poslova novima. Alat koristi knjižnicu, a služi korištenju iz naredbenog retka te zadovoljava POSIX specifikaciju.

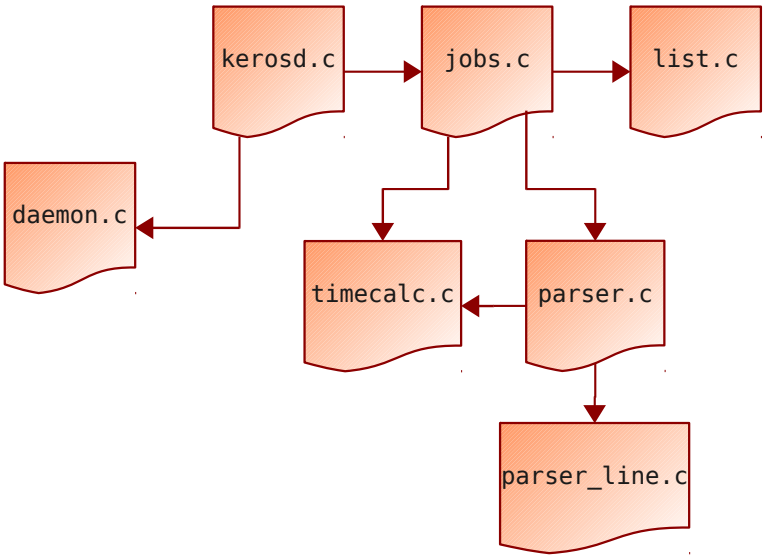
Svaka se od te tri cjeline nalazi u zasebnom kazalu unutar kazala `src/`. Iz sljedećeg je ispisa vidljiva spomenuta hijerarhija.

```
$ ls src/
common  daemon  lib  Makefile.am  util
```

U kazalu `common/` nalazi se samo datoteka `common.h` u kojoj su zapisane neke konstantne koje su zajedničke za sve cjeline. Kazalo `daemon/` sadrži programski kôd pozadinskog procesa, kazalo `lib/` sadrži kôd knjižnice, dok se u kazalu `util/` nalazi kôd alata naredbenog retka. Datoteka `Makefile.am` potrebna je radi GNU razvojnog sustava.

Kôd je organiziran po modulima. Organizacija koda pozadinskog procesa može se vidjeti na slici 4.1.

U nastavku slijedi opis funkcionalnosti pozadinskog procesa po modulima.



Slika 4.1: Organizacija koda pozadinskog procesa po modulima

`kerosd.c`: modul u kojem se nalazi funkcija `main()`. Nakon što se pokrene, program provjerava parametre naredbenog retka te se, ako je tako određeno, prebacuje u pozadinu i odvaja od terminala (tj. postaje pozadinski proces). Zatim maskira sljedeće signale: `SIGCHLD` (tako da može ispravno pričekati procese koji se stvaraju kad se posao izvršava) te `SIGINT`, `SIGQUIT` i `SIGTERM` (ovi se signali hvataju kako bi se zabilježilo vrijeme završetka programa što je potrebno radi ostvarenja funkcionalnosti slične programu `anacron`). Nakon toga se učitavaju poslovi svih korisnika te se na kraju pokreće glavni dio programa u kojem se pokreću poslovi.

`daemon.c`: ovaj modul sadrži samo jednu funkciju (`daemonize()`), koja služi prebacivanju programa, koji poziva ovu funkciju u pozadinu na siguran način.

`jobs.c`: ovaj modul sadrži dvije bitne funkcije. Prva je `load_jobs()` koja učitava sustavske poslove (iz datoteke `/etc/kerostab`) i poslove ostalih korisnika (iz kazala `/var/spool/keros/kerostabs/`). Ukoliko se ni jedna datoteka s poslovima ne uspije pročitati, program završava s radom. Druga je funkcija `run_jobs()`. Funkcija u beskonačnoj petlji uzima posao s vrha liste poslova (koja je razvrstana, a na vrhu se nalazi posao koji se treba prvi izvršiti), izračunava razliku između vremena kad se posao treba izvršiti i trenutnog vremena te zatim „odspava” toliko vremena. Poslije toga izvršava taj posao te ga ponovno umeće u listu (poslovi se ne izvršavaju samo jednom već periodički).

`list.c`: sadrži funkcije kojima je ostvarena razvrstana lista. Bitno je naglasiti kako je razvrstanu listu jednostavno moguće zamijeniti nekom drugom strukturom podataka. U tom bi slučaju trebalo napisati funkcije za brisanje i dodavanje elemenata u tu strukturu te promijeniti tip podataka koje struktura drži u svojim čvorovima.

`timecalc.c`: najsloženiji dio programa. Na temelju vremenske oznake posla izračunava prvo vrijeme u budućnosti kada se posao treba izvršiti. Pri tome referentno vrijeme može biti proizvoljno: obično se uzima trenutno vrijeme, no prilikom ostvarenja funkcionalnosti slične programu `anacron` kao referentno se vrijeme uzima vrijeme posljednjeg završetka programa.

`parser.c`: u ovom se modulu raščlanjuje sintaksa (engl. *parse*) ulazne datoteke s poslovima. Raščlamba se ne radi unutar retka, nego se iz datoteke izdvajaju retci koji odgovaraju poslovima, a poslovi se šalju na raščlambu u modul `parser_line.c`.

`parser_line.c`: u ovom se modulu poslovi raščlanjuju te se iz zapisa u datoteci s poslovima (koji je tekstualni pa ga je jednostavno čitati i uređivati) stvara struktura koja je pogodna za daljnje korištenje u programu.

Uz module, koji su dio pozadinskog procesa, postoje još dva modula, koji su dijelovi knjižnice, odnosno alata. U nastavku slijedi opis tih modula.

`keros.c`: glavni modul knjižnice. Sadrži funkcije koje drugi programi mogu koristiti za upravljanje poslovima. Primjer je programa koji koristi knjižnicu alat `kerostab`.

`kerostab.c`: glavni modul alata. Raščlanjuje argumente naredbenog retka te u ovisnosti o njima ispisuje poslove korisnika koji je alat pozvao, briše poslove ili ih stvara.

4.2. POSIX specifikacija

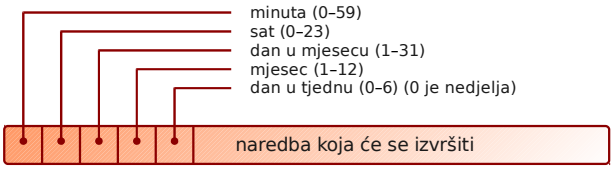
Specifikacija programa `crontab` dio je standarda *IEEE Std 1003.1*, poznatog i pod nazivom POSIX, od drugog izdanja. Specifikacija opisuje sučelje alata naredbenog retka (`crontab`), okolinu u kojoj se pokreću poslovi te oblik poslova programa `cron`.

4.2.1. Oblik datoteke s poslovima

Poslovi programa `cron` tipično se nalaze u datoteci po imenu `crontab` koja se nalazi u kazalu `/etc/`. To je sustavska `crontab` datoteka i nju može uređivati samo korisnik *root*. Usto, korisnici imaju svoje `crontab` datoteke, no u slučaju *Vixie Cron* inačice, iz

sigurnosnih razloga, ne mogu ih ručno uređivati već se za to koristi program imena `crontab`.

Svaki je posao predstavljen jednim retkom teksta u `crontab` datoteci. Svaki se redak, dakle posao, sastoji od šest polja odvojenih razmacima (engl. *blank characters*). Od toga, prvih pet polja služe za opis vremenske oznake, a posljednje polje je naredba koja se izvršava zajedno sa svojim parametrima. Polja zajedno s pripadnim opisima prikazana su na slici 4.2.



Slika 4.2: Popis polja cron posla

Program ostvaren u okviru ovog rada podržava i oblik posla koji ima sedam polja. Pri tome se taj oblik koristi samo za sustavske poslove (tj. za poslove u datoteci `/etc/kerostab`), dok se za korisničke poslove koristi standardni oblik posla. Kod poslova sa sedam polja, prvih pet polja i dalje služi za opis vremenske oznake, a naredba se sad nalazi u sedmom polju. U petom se polju nalazi ime korisnika s čijim se pravima posao treba pokrenuti. To omogućava korisniku *root* (koji jedini može uređivati sustavske poslove) dodavanje poslova drugih korisnika.

Svako od pet polja vremenske oznake može biti zvjezdica (`*`, označava sve dozvoljene vrijednosti), element ili niz elemenata odvojenih zarezom. Element se sastoji ili od jednog broja ili od dva broja odvojenih crticom (`-`, označava raspon vrijednosti koji uključuje obje granice).

Vremenskom se oznakom opisuje vrijeme kada se posao treba pokrenuti: kad god trenutno vrijeme na operacijskom sustavu zadovoljava vremensku oznaku posla, posao se treba izvršiti. Razlučivost oznake je jedna minuta – ne može se odrediti trenutak unutar odabrane minute kada će se posao izvršiti, to ovisi o ostvarenju. Dan se unutar oznake može opisati preko dva polja: dan u mjesecu i dan u tjednu. Podudarajući se dani iz vremenske oznake određuju na sljedeći način:

- ako se u poljima za mjesec, dan u mjesecu i dan u tjednu nalaze elementi ili liste, tada su podudarajući dani određeni tim poljima;

- ako se u poljima za mjesec, dan u mjesecu i dan u tjednu nalaze zvjezdice, tada se svaki dan treba podudarati;
- ako se u polju za mjesec i/ili u polju za dan u mjesecu nalazi element ili lista, a u polju za dan u tjednu je zvjezdica, tada su podudarajući dani određeni poljima za mjesec i dan u mjesecu;
- ako se u poljima za mjesec i za dan u mjesecu nalaze zvjezdice, a u polju za dan u tjednu je element ili lista, tada se samo dani iz polja za dan u tjednu trebaju podudarati;
- ako se ili u polju za mjesec ili u polju za dan u mjesecu nalazi element ili lista, a u polju za dan u tjednu je također element ili lista, tada su podudarajući svi dani kod kojih se podudara ili mjesec ili dan u mjesecu te dan u tjednu.

Slično, vrijeme unutar dana kada se posao treba izvršiti određeno je poljima za minute i sate. Ukoliko se u oba polja nalaze zvjezdice, tada se svaka minuta i svaki sat podudaraju.

Šesto polje svakog posla niz je znakova koji predstavljaju naredbu. Znak postotka u ovom polju koristi se kao oznaka novog retka (`\n`), a bilo koji znak ispred kojeg se nalazi obrnuta kosa crta (\) se tumači doslovno (to uključuje i znak postotka). Dio niza do prvog znaka postotka (ili do kraja retka) uzima se kao naredba koja se izvršava u ljusci; eventualni ostatak retka se šalje naredbi kao standardni ulaz.

Prazni retci te retci u kojima je znak „#” prvi znak koji nije razmak se ignoriraju.

Svaki korisnik ima svoju datoteku s poslovima te se poslovi nekog korisnika izvršavaju s pravima korisnika koji je vlasnik datoteke.

U tablici 4.1 nalazi se nekoliko primjera poslova programa `cron`.

Tablica 4.1: Primjeri poslova programa cron

Posao	Objašnjenje
`0 0 25 12 * mailx ivan%Sretan Božić!`	Slanje Božićne čestitke.
`0 3 * * * rm -rf /tmp/*`	Brisanje sadržaja privremenog kazala svako jutro u 3 sata.
`0 14 * * 1-5 echo "Plati parkiranje!" | gammu sendsms text 0971234567`	Šalje SMS opomenu za parkiranje svaki radni dan u 14:00.
`23 59 12 * 4 /sbin/halt`	Gasi računalo jednu minutu prije svakog petka trinaestog.

4.3. Dodatni zahtjevi

4.3.1. Sigurnosni zahtjevi

Kod ovakvih programa potrebno je posvetiti posebnu pažnju sigurnosti. Najveći problem predstavlja činjenica kako se takav program izvršava s pravima korisnika *root*, pa treba paziti da se poslovi običnih korisnikâ ne izvršavaju s ovlastima tog korisnika. Time bi programi, koji se pokreću, mogli ugroziti sigurnost cijelog sustava – što zbog nehotične pogrješke, što zbog mogućeg zlonamjernog koda. U nastavku slijede primjeri kako pridonijeti sigurnosti kod ovakvih programa.

Izvršavanje poslova

Tipično, samo neki poslovi pripadaju korisniku *root*. Svi ostali pripadaju neprivilegiranim korisnicima. Prilikom pokretanja posla koristi se tehnika dvostrukog stvaranja procesa i izvršavanja (engl. *double fork and exec*). Kako se glavni program pokreće s pravima korisnika *root*, glavni program mora promijeniti efektivni identifikator grupe (EGID) te efektivni identifikator korisnika (EUID) nakon stvaranja novog procesa, a prije izvršavanja programa. U takve se svrhe koriste sustavski pozivi `setgid()` te `setuid()`, koji kao jedini argument primaju identifikator grupe, odnosno korisnika, koji će se postaviti kao efektivni identifikator.

Zaštita poslova

Ranije izvedbe programa `cron` uzimale su datoteke s poslovima iz početnih kazala korisnikâ. To rješenje ima jedan nedostatak: prilikom čitanja poslova te se datoteke moraju potražiti u svim početnim kazalima, i u onima gdje ona postoji i u onima gdje ne postoji (a ne zna se unaprijed koji korisnici imaju poslove, a koji ih nemaju). Zato kasnije inačice `cron`a sve poslove uzimaju iz jednog kazala (engl. *spool area*). Ne primijene li se određene mjere i taj pristup ima neke nedostatke: svi korisnici mogu vidjeti koji korisnici imaju poslove, a koji ne (zadiranje u privatnost); svi korisnici mogu stvarati datoteke s poslovima (s obzirom na to kako naziv datoteke određuje vlasnika posla, to znači da bilo koji korisnik može napraviti poslove u ime drugog korisnika).

Nedostatci se mogu riješiti korištenjem klasičnog sustava dopuštenja i nekih dodatnih oznaka (postavljanjem tzv. *sticky* bita na kazalo te *setgid* bita na program).

Zamisao je korisnicima onemogućiti izravno stvaranje poslova te im pružiti posebni program za tu namjenu (u izvornom ostvarenju riječ je o alatu `crontab`). Taj program stvara datoteku s poslovima, a naziv datoteke određuje prema identifikatoru korisnika koji ga pokreće. Program mora imati dopuštenje za stvaranje datoteka u kazalu s poslovima, ali korisnik, koji pokreće program (kao ni ostali korisnici), ne smije moći ni čitati sadržaj tog kazala ni imati mogućnost ulaska u to kazalo. To se ostvaruje stvaranjem nove grupe, pridruživanjem te grupe programu i kazalu te postavljanjem odgovarajućih dopuštenja. Također, vlasnik svih izvršnih datoteka i kazala s poslovima treba biti korisnik *root*.

```
# chown root /tmp/kerostest/bin/*
# chown -R root /tmp/kerostabs/
# addgroup kerostab
Adding group `kerostab' (GID 1003) ...
Done.
# chgrp kerostab /tmp/kerostabs/
# chmod u=rwx,g=wx,o= /tmp/kerostabs/
```

Nakon izvršavanja ovih naredbi, obični korisnici (tj. korisnici koji nisu korisnik *root* i ne pripadaju grupi *kerostab*) ne mogu čitati, stvarati ni pregledavati svoje, ni poslove drugih korisnika. S druge strane korisnici, koji su u grupi *kerostab*, mogu stvarati i pregledavati svoje, ali i poslove drugih korisnika. Zato se poslovi stvaraju pomoću posebnog alata (u slučaju ovog ostvarenja riječ je o programu `kerostab`). Kako bi taj program (kojeg pokreću obični, neprivilegirani, korisnici) mogao stvarati poslove, pridružuje mu se ista grupa kojoj pripada i kazalo s poslovima te mu se postavlja zastavica kojom se označava da se program treba pokrenuti s pravima grupe kojoj pripada:

```
# chgrp kerostab /tmp/kerostest/bin/kerostab
# chmod u=rwx,g=r,o=rx /tmp/kerostest/bin/kerostab
# chmod g+s /tmp/kerostest/bin/kerostab
```

Budući da kazalo s poslovima dozvoljava korisnicima, koji su u grupi *kerostab*, pregledavanje i stvaranje poslova time se postiže željeni učinak: obični korisnici ne mogu izravno pregledavati ni stvarati poslove, ali koristeći program `kerostab` to mogu.

4.3.2. Interval pokretanja

Svaki posao programa `cron` ima svoju vremensku oznaku kojom je određeno kada će se on izvršiti. No, problem nastaje kada se posao treba izvršiti, a računalo nije pokrenuto. U tom slučaju posao se neće izvršiti. Postoji više načina kako ublažiti posljedice toga. Kod jednog se načina uvodi interval unutar kojeg se posao može pokrenuti. Umjesto da se posao pokrene u točno određeno vrijeme (npr. svaki dan u 10 sati), posao se napravi tako da se može pokrenuti u nekom vremenskom rasponu (primjerice svaki dan između 9 i 11 sati), ali samo jedanput. Kod drugog načina se vrijeme pokretanja određuje relativno, u odnosu na trenutak pokretanja sustava; primjerice 20 minuta nakon pokretanja sustava.

Program ostvaren u okviru ovog rada taj problem rješava na način da prilikom pokretanja pokreće dodatno jednom sve poslove koji su se trebali pokrenuti dok je sustav bio ugašen. Kako bi to mogao, prilikom gašenja program sprema trenutno vrijeme na sustavu (što je zapravo vrijeme od kojeg će se početi tražiti propušteni poslovi prilikom sljedećeg pokretanja programa).

4.3.3. Ostali zahtjevi

Uz navedene zahtjeve, postoji još čitav niz drugih zahtjeva, od kojih su neki važniji od drugih. Jedan je od važnijih zahtjeva uzimanje u obzir mijenjanje vremena. Naime, ukoliko se vrijeme na sustavu pomakne unaprijed (primjerice pomak od jednog sata u proljeće), svi poslovi, koji bi se trebali pokrenuti u preskočenom satu, neće se izvršiti ako se ne vodi računa o tome. Slično tome, ako se sat vraća unatrag, potrebno je paziti da se poslovi, koji se izvršavaju u ponovljenom satu, ne izvrše dvaput.

U ostvarenju vrijeme se sprema u varijable tipa `time_t` te `struct tm`. Tip `time_t` sadrži broj sekundi od 1. 1. 1970. te zauzima malo prostora (riječ je od 32-bitovnom broju). S druge strane, u tip `struct tm` se sprema tzv. razlomljeno vrijeme: struktura se sastoji od varijabli (cjelobrojnog tipa) za sekunde, minute, sate, dan u mjesecu itd. U ostvarenju se za pretvorbu razlomljenog vremena u podatak tipa `time_t` koristi funkcija `mktime()`. Posebnost je te funkcije što vodi računa o ljetnom/zimskom računanju vremena pa je ranije spomenuti zahtjev inherentno riješen.

Dijagnostički ispis jedna je funkcionalno nebitna stvar, no treba biti sastavnim dijelom jednog ovakvog programa. S obzirom na to da se glavni program pokreće u pozadini, ne može se koristiti uobičajeni ispis koristeći funkciju `printf()` (jedna od stvari koja se prilikom pokretanja programa u pozadini radi je odvajanje od terminala). Ostvarenje zato koristi program *syslog* koji sve poruke ispisuje u dnevnik (obično je riječ o datoteci `/var/log/syslog`).

Još jedan zahtjev mogao bi biti da se poslovi koji se u nekoj minuti moraju pokrenuti ne pokreću istodobno, budući da se time može izazvati zagušenje računala. Ovdje će se predstaviti dva moguća rješenja tog problema. Prvo je moguće rješenje da se poslovi ne pokreću u isto vrijeme već da se za svaki posao, koji se te minute treba pokrenuti, nasumično odredi točno vrijeme unutar te minute kad će se pokrenuti. Time bi se poslovi raspršili i smanjila bi se vjerojatnost velikog zagušenja računala. Drugo je moguće rješenje da se odredi određena vrijednost opterećenja procesora te da se poslovi izvršavaju, samo ako je opterećenje procesora, u trenutku kad bi se posao trebao pokrenuti, manje od te unaprijed određene vrijednosti. Ako je opterećenje procesora preveliko, pokretanje posla bi se odgodilo za neko kratko vrijeme (primjerice za dvije minute).

Oba prijedloga imaju jedan veliki nedostatak: posao se ne izvršava točno onda kad ga je korisnik zakazao, što može imati neželjene posljedice ukoliko je vrijeme izvršavanja jako bitno. Iz tog razloga funkcionalnost, koja bi zadovoljila taj zahtjev, nije dodana u ostvarenje.

4.4. Načelo rada ostvarenja

Ostvarenje je oblikovano tako da radi po poboljšanom algoritmu kako je to opisano u poglavlju 2.1. Ipak, postoje neke razlike između tog algoritma i algoritma, koji se koristi u ostvarenju: datoteke s poslovima ne nalaze se u početnim kazalima korisnikâ, već u jednom kazalu; ne koristi se Franta-Malyjeva lista događaja, već obična razvrstana lista; mogu se izvršavati poslovi koji su propušteni za vrijeme dok program nije radio (npr. dok je računalo bilo isključeno).

Algoritam po kojem ostvarenje radi slijedi u nastavku.

1. Prilikom pokretanja, pokušaj otvoriti datoteku imena `.timestamp` u kazalu s poslovima:
    1. ako datoteka postoji, pročitaj iz nje vrijeme kad je program posljednji put radio;
    2. postavi zastavicu „pokreni propuštene poslove”;
2. pretraži datoteku sa sustavskim poslovima te sve datoteke u kazalu s poslovima;
3. za svaku pronađenu datoteku i za svaki posao u njoj:
    1. ako je zastavica „pokreni propuštene poslove” postavljena:
        1. postavi referentno vrijeme na vrijeme kad je program posljednji put radio;
        2. izračunaj prvo sljedeće vrijeme nakon referentnog vremena kad se posao treba izvršiti;
        3. ako je to vrijeme u prošlosti:
            1. postavi za taj posao zastavicu „pokreni samo jednom”;
            2. zakaži posao sljedeće minute i dodaj ga u listu poslova;
    2. odredi sljedeće vrijeme u budućnosti kad se posao treba izvršiti;
    3. dodaj posao u listu poslova;
4. uđi u glavnu petlju:
    1. pogledaj posao na početku liste te odredi za koliko se vremena treba izvršiti;
    2. odspavaj tu količinu vremena;
    3. nakon buđenja, izvrši u pozadini program s početka liste s povlasticama korisnika kojem taj posao pripada;
    4. ako je za taj posao postavljena zastavica „pokreni samo jednom”:
        1. izbriši posao iz liste;
    5. inače:
        1. ažuriraj vrijeme sljedećeg izvršavanja tog posla te ga, tako izmijenjenog, premjesti na odgovarajuće mjesto u listi događaja.

Algoritam je dosta složeniji od onog opisanog u poglavlju 2.1, uglavnom zbog toga što se pokreću poslovi koju su se trebali pokrenuti u vremenu dok program nije radio.

Takvi se poslovi pokreću samo jednom, tj. ne onoliko puta koliko bi se inače pokrenuli da je program bio pokrenut.

4.5. Korištenje ostvarenja

U nastavku slijedi opis postupka dohvaćanja, izgradnje te korištenja programa ostvarenog u okviru ovog rada.

4.5.1. Dohvaćanje i izgradnja programskog koda

Tijekom razvoja ostvarenja koristi se sustav za upravljanje revizijama Git. Programski se kôd sprema na poslužitelj sustava Gitorious. Gitorious je naziv internetskog domaćina za projekte otvorenog koda temeljene na suradnji (engl. *collaborative development*). Korisnicima pruža skladišni prostor za programski kôd kojim se može upravljati koristeći Git.

Ukoliko se želi dohvatiti programski kôd ostvarenja, potrebno je klonirati Git repozitorij u kojem se on nalazi. To se radi na sljedeći način:

```
$ git clone git://gitorious.org/kruno/keros.git keros
Initialized empty Git repository in /tmp/git/keros/.git/
remote: Counting objects: 713, done.
remote: Compressing objects: 100% (696/696), done.
remote: Total 713 (delta 465), reused 0 (delta 0)
Receiving objects: 100% (713/713), 1.34 MiB | 56 KiB/s, done.
Resolving deltas: 100% (465/465), done.
```

Nakon što kloniranje uspije, pojavljuje se novo kazalo `keros/`, u kojem se nalazi projekt, koji se sada može na uobičajen način izgraditi. Skripta `configure` prima razne standardne parametre, a najčešći je parametar `--prefix`, kojim se može promijeniti odredište gdje će se programi, knjižnice i ostale datoteke, koje pripadaju projektu instalirati. Ako se taj parametar izostavi, podrazumijeva se odredište `/usr/local/`. Primjera radi, kao odredište će se postaviti kazalo `/tmp/keros/`.

```
$ ./configure --prefix=/tmp/keros/
```

Nakon uspješnog podešavanja projekta za sustav, na kojem se izgrađuje, projekt se može izgraditi:

```
$ make
```

Konačno, projekt se može instalirati:

```
$ make install
```

Nakon ovog posljednjeg koraka može se provjeriti jesu li sve datoteke instalirane:

```
$ cd /tmp/keros/
$ ls *
bin:
kerosd  kerostab

include:
keros.h

lib:
libkeros.a
```

Iz ovog se ispisa može vidjeti kako se u potkazalu `bin/` nalaze programi (pozadinski proces `kerosd` te alat `kerostab`), u potkazalu `lib/` statička knjižnica `libkeros.a`, a u potkazalu `include/` zaglavlje potrebno za razvoj programa koji koriste knjižnicu.

4.5.2. Korištenje programa

Program zahtijeva postojanje barem jedne datoteke s poslovima prilikom pokretanja. Osim korisničkih poslova, postoje i sustavski poslovi koji se nalaze u datoteci `/etc/kerostab`. Nakon instalacije programa potrebno je napraviti kazalo gdje se nalaze korisnički poslovi, a koje je to točno kazalo ovisi o konstanti `CRON_SPOOL_AREA` definiranoj u datoteci `common/common.h`. Trenutno je to kazalo `/tmp/crontabs/`.

```
$ mkdir /tmp/kerostabs
```

Za potrebe ispitivanja možemo napraviti sljedeći posao:

```
$ /tmp/keros/bin/kerostab
* * * * * date +\%H:\%M.\%S > /tmp/vrijeme_pokretanja
^D
```

Taj se posao izvršava svake minute, a naredba koja se izvršava sastoji se od ispisivanja trenutnog vremena (sat, minuta i sekunda) i od preusmjeravanja tog ispisa u datoteku `/tmp/vrijeme_pokretanja`. Može se vidjeti kako je znak postotka predznačen s obrnutom kosom crtom što znači da se on shvaća doslovno prilikom izvršavanja naredbe. Kada ne bi bio predznačen, sve dalje od prvog znaka postotka bi se shvatilo kao niz koji se treba naredbi prije tog znaka predati kao standardni ulaz.

Posao se u gornjem primjeru dodao kao standardni ulaz naredbi `kerostab`. Nakon pozivanja te naredbe bez parametara, poslovi se unose redom, po jedan u svakom retku. Kad su uneseni svi poslovi, unosi se znak za kraj retka u praznom retku (obično je to znak `^D` koji se dobiva istovremenim pritiskom tipki *Control* i slova D). U ovome slučaju datoteka s poslovima još ne postoji, pa će se stvoriti i to unutar kazala `/tmp/kerostabs/`, a naziv te datoteke biti će jednak korisničkom imenu. Popis dopuštenih argumenata naredbe `kerostab` dan je u tablici 4.2.

Tablica 4.2: Popis dopuštenih argumenata naredbe kerostab

Argument	Objašnjenje
-l (malo slovo L)	Ispisuje poslove korisnika koji je pozvao naredbu.
-r	Briše sve poslove korisnika koji je pozvao naredbu.
-h	Ispisuje poruku pomoći.

Dodatno, ako se kao jedini argument pošalje datoteka, tada će se svi korisnikovi poslovi zamijeniti poslovima koji se nalaze u toj datoteci.

Nakon što postoji barem jedna datoteka s poslovima, može se pokrenuti glavni program (`kerosd`). Argumenti koje program prima popisani su u tablici 4.3.

Tablica 4.3: Popis dopuštenih argumenata programa kerosd

Argument	Objašnjenje
-d	Pokreće program u pozadini.
-v	Ispisuje trenutnu inačicu programa.
-h	Ispisuje poruku pomoći.

Uobičajeno je program pokrenuti u pozadini:

```
$ /tmp/keros/bin/kerosd -d
```

Program ispisuje sve dijagnostičke poruke koristeći program *syslog*. Iz sljedećeg su ispisa vidljive poruke koje program ispisuje nakon pokretanja.

```
$ tail -f /var/log/syslog
Jun  5 16:04:56 perseus keros[20444]: Starting...
Jun  5 16:04:56 perseus keros[20444]: jobs.c:273: Cannot opet timestamp
file.
Jun  5 16:04:56 perseus keros[20444]: parser.c:245: Cannot parse crontab
/etc/kerostab
Jun  5 16:04:56 perseus keros[20444]: Initial crontabs loaded.
Jun  5 16:04:56 perseus keros[20444]: Listing all jobs, sorted by time.
Jun  5 16:04:56 perseus keros[20444]: JOB 0: time:  1275746700 (16:05;
5/06/2010; Sat)
Jun  5 16:04:56 perseus keros[20444]:        uid:   1000
Jun  5 16:04:56 perseus keros[20444]:        cmd:   date +%H:%M.%S >
/tmp/vrijeme_pokretanja
Jun  5 16:04:56 perseus keros[20444]:        stdin: (null)
Jun  5 16:04:56 perseus keros[20444]: Running jobs in queue.
Jun  5 16:04:56 perseus keros[20444]: Sleeping for 4 seconds ;)
```

Iz ispisa se može vidjeti kako program nije uspio pročitati datoteku u kojoj je zapisano vrijeme kad je posljednji put bio pokrenut (engl. *time-stamp file*). Ta je datoteka potrebna kako bi se mogli pokrenuti poslovi koji se nisu izvršili za vrijeme dok je program bio ugašen (to je posao koji inače radi program `anacron`). S obzirom na to da se program dosad nije pokretao, ta datoteka ne postoji – stvorit će se kada program završi s radom. Dalje se može vidjeti da program nije pronašao sustavsku datoteku s poslovima (`/etc/kerostab`), što je u redu jer ne postoji.

Nakon što program učita sve datoteke s poslovima, ispisuje pronađene poslove. U ovom primjeru postoji samo jedan posao i za njega se može vidjeti sljedeće:

- prvo sljedeće vrijeme kad se posao treba izvršiti je 1 275 746 700 (Unix vrijeme, broj sekundi od 1. 1. 1970.), što je zapravo subota, 5. 6. 2010. u 16:05 (prva puna minuta nakon pokretanja programa što je, zaista, točno budući da se program treba izvršavati svake minute);
- identifikator korisnika s čijim će se pravima posao izvršiti je 1000 – identifikator se određuje na osnovu naziva datoteke (koji je jednak korisničkom imenu) koristeći standardnu POSIX funkciju `getpwnam()`;
- naredba koja će se izvršiti je `date +%H:%M.%S > /tmp/vrijeme_pokretanja` – tu se može vidjeti kako više nema obrnutih kosih crta ispred znaka postotka;
- niz koji se predaje kao standardni ulaz naredbi je `NULL`, tj. ne postoji.

Program zatim ispisuje kako „spava” sljedeće četiri sekunde: kako se iz ranijeg ispisa moglo vidjeti, posao se treba izvršiti u 16:05:00 (razlučivost je vremena jedna minuta, program `kerosd` poslove pokreće prve sekunde pune minute), a vrijeme

pokretanja programa bilo je 16:04:56 što znači da program, zaista, treba pokrenuti posao za četiri sekunde.

Kad je program odspavao te četiri sekunde, pokreće se posao. Program ispisuje sljedeće:

```
Jun  5 16:05:00 perseus keros[20448]: keros[20448]: (kruno) CMD (date +%H:
%M.%S > /tmp/vrijeme_pokretanja)
```

Vidi se da se posao pokreće u 16:05:00, korisnik s čijim se pravima izvršava je *kruno*, a naredba je `date +%H:%M.%S > /tmp/vrijeme_pokretanja`. Može se provjeriti je li se posao stvarno pokrenuo: ako je, tada se u datoteci `/tmp/vrijeme_pokretanja` treba nalaziti točno vrijeme kad se posao pokrenuo.

```
$ cat /tmp/vrijeme_pokretanja
16:05.00
```

Dakle, posao se pokrenuo. Nakon što ga je program pokrenuo, posao se ponovno umeće u popis poslova (s ažuriranim vremenom pokretanja posla), a program opet uzima posao s vrha liste te sve kreće isponova. Kako postoji samo jedan posao, na vrhu liste se opet nalazi isti posao. Program ispisuje:

```
Jun  5 16:05:00 perseus keros[20444]: Sleeping for 60 seconds ;)
```

Dakle, posao će se ponovno izvršiti za 60 sekundi (jednu minutu) što je u redu jer se, kako je već spomenuto, posao izvršava svake minute. Konačno, kad program završava s radom, zapisuje vremensku markicu u datoteku `/tmp/kerostabs/.timestamp`:

```
Jun  5 16:06:21 perseus keros[20444]: Caught signal 15, dumping timestamp
(1275746781).
```

Može se provjeriti je li markica ispravno zapisana:

```
$ od -i /tmp/kerostabs/.timestamp
0000000  1275746781
$ date -d @1275746781
Sat Jun  5 16:06:21 CEST 2010
```

Može se vidjeti da vremenska markica odgovara poruci ispisanoj pomoću programa *syslog*.

4.5.3. Korištenje knjižnice

Sučelje knjižnice programa *keros* pruža vanjskim programima jednu strukturu i četiri funkcije. Programi, koji koriste knjižnicu, trebaju uključiti pripadno zaglavlje `keros.h`. U nastavku slijedi opis strukture i funkcija.

```
struct keros_job {...};
```

Struktura (čvor liste), koja se koristi zajedno s funkcijama iz knjižnice, sastoji se od posla (tekstualni oblik, redak iz datoteke s poslovima) te od pokazivača na sljedeću takvu strukturu.

```
int keros_get_users_jobs(uid_t uid, struct keros_job **head);
```

Funkcija koja popisuje poslove određenog korisnika. Argumenti funkcije su identifikator korisnika, čiji se poslovi popisuju, te glava liste, u koju se poslovi spremaju. Sva potrebna memorija osigurava se unutar funkcije. Funkcija vraća 0 ako je uspješno popisala poslove, 1 ako datoteka s poslovima ne postoji ili se ne može otvoriti te -1 u slučaju pogrješaka.

```
void keros_delete_list(struct keros_job *head);
```

Oslobađa memoriju rezerviranu za čvorove liste tijekom izvođenja funkcije `keros_get_users_jobs()`. Funkcija kao jedini argument prima pokazivač na glavu liste.

```
int keros_remove_users_jobs(uid_t uid);
```

Pozivanjem ove funkcije brišu se svi poslovi korisnika s identifikatorom `uid`. Vraća 0 ako su uspješno izbrisani poslovi korisnika, a -1 u slučaju pogrješaka.

```
int keros_replace_jobs(uid_t uid, FILE *infile);
```

Ova funkcija mijenja trenutne poslove korisnika s identifikatorom `uid` s onima u datoteci na koju pokazuje `infile`. Funkcija vraća 0 ako je uspjela zamijeniti poslove, a -1 u slučaju pogrješaka.

U sljedećem se ispisu nalazi izvorni kôd programa koji koristi funkcije iz knjižnice za ispis poslova korisnika koji poziva program.

```
$ cat kerostest.c
#include <stdio.h>      /* printf() */
#include <stdlib.h>     /* EXIT_SUCCESS, EXIT_FAILURE */
#include "keros.h"      /* Zaglavlje knjižnice. */

int main(int argc, char *argv[])
{
    int retval;
    uid_t uid;
    struct keros_job *job, *head;

    uid = getuid();
    retval = keros_get_users_jobs(uid, &head);
    if (retval == -1) {
        fprintf(stderr, "Ne mogu dohvatiti korisnikove poslove\n");
        return EXIT_FAILURE;
    } else if (retval == 1)
        printf("Korisnik nema poslova!\n");
    else {
        for (job=head; job != NULL; job=job->next)
            printf("%s\n", job->job);
    }

    return EXIT_SUCCESS;
}
```

Program na početku poziva sustavski poziv `getuid()` kako bi dohvatio identifikator korisnika koji poziva program. Zatim poziva funkciju `keros_get_users_jobs()` iz knjižnice prenoseći kroz parametre identifikator korisnika te glavu liste te provjerava povratnu vrijednost. Ako se dogodila pogrješka tijekom dohvaćanja poslova, program izlazi vraćajući kôd pogrješke (koristi se standardna konstanta `EXIT_FAILURE`). Ako datoteka s poslovima ne postoji, to se ne smatra grješkom, pa program izlazi vraćajući kôd uspjeha (konstanta `EXIT_SUCCESS`). Naposljetku se, ukoliko je funkcija uspješno dohvatila poslove, ispisuju poslovi na standardni izlaz te program uspješno izlazi.

Program se prevodi na sljedeći način:

```
$ gcc -I/tmp/keros/include/ -L/tmp/keros/lib/ -o kerostest kerostest.c
-lkeros
```

Prevodiocu je potrebno „reći” gdje se nalazi zaglavlje knjižnice (odrednica `-I`), gdje se nalazi knjižnica (odrednica `-L`) te kako se knjižnica zove, kako je to opisano u poglavlju 3.2 (odrednica `-l`). Prevođenjem nastaje izvršna datoteka `kerostest` čijim pokretanjem dobivamo ispis poslova – ispis je jednak onome koji se dobije pokretanjem alata s opcijom `-l` (`kerostab -l`).

```
$ ./kerostest
* * * * * date +\%H:\%M.\%S > /tmp/vrijeme_pokretanja
```

5. Zaključak

Kao što je napisano u uvodu, cilj je ovog rada je proučiti teorijsku podlogu za izradbu programa za vremenski odgođeno pokretanje drugih programa te napraviti jedan takav program. Razmotreni su postupci, zahtjevi, ali i ostali alati i tehnologije potrebne za izradbu takvog programa.

Nakon provedenog istraživanja, stekao sam dojam o složenosti izvedbe takvog programa i upoznao se s alatima koji olakšavaju izradbu i održavanje programskog projekta. Također, naučio sam koristiti se dijeljenim knjižnicama.

Nakon proučavanja postojećih rješenja, mogu reći kako se na GNU/Linux operacijskom sustavu koristi staro i provjereno ostvarenje (*Vixie Cron*) koje ima veliki nedostatak – ne pruža moderne značajke. Ne definira programsko sučelje, pa ga programi ne mogu na jednostavan način koristiti. Primjerice, niti jedan program za planiranje vremena na spomenutom sustavu ne koristi `cron` za raspoređivanje svojih zadataka; poput popisa zadataka (engl. *to-do list*). Umjesto toga, jedan tipični takav program ima svoje ostvarenje za tu namjenu što povlači za sobom činjenicu da mora biti pokrenuti cijelo vrijeme kako bi se zadatci tog programa uopće mogli izvršiti. Kada bi taj program koristio `cron`, on ne bi morao biti pokrenut, već bi ga `cron` pokretao po potrebi.

Ostvarenje izrađeno u okviru ovog rada temelji se na specifikaciji iz POSIX norme. Poslovi su organizirani na sličan način, oblik poslova slijedi specifikaciju, a alat naredbenog retka radi na isti način i podržava odrednice koje su opisane u specifikaciji. Značajke koje ovo ostvarenje ima, a *Vixie Cron* nema su programsko sučelje te mogućnost pokretanja poslova koji su se trebali pokrenuti za vrijeme dok je računalo bilo ugašeno.

Nisu svi detalji POSIX specifikacije ostvareni. Dvije bitne značajke koje nedostaju su: ne postavljaju se varijable okoline posla (iako je započet rad na tome) i poslovima se ne prenosi standardni ulaz.

6. Literatura

1. IEEE, The Open Group: crontab, URL: http://www.opengroup.org/onlinepubs/9699919799/utilities/crontab.html
2. IEEE, The Open Group: at, URL: http://www.opengroup.org/onlinepubs/9699919799/utilities/at.html
3. IEEE, The Open Group: The Open Group Base Specifications Issue 6, URL: http://www.opengroup.org/onlinepubs/009695399/
4. Vixie, P. Stranice priručnika programa Vixie Cron. 4th Berkeley Distribution, 2006.
5. Koenig, T. Stranice priručnika programa at. Linux Programmer's Manual, 1996.
6. W. R. Franta, K. Maly. An efficient data structure for the simulation event set. Communications of the ACM, 1977., 8, 596-602.
7. Gattol, M.: Time - An important Subject with any OS, URL: http://sunoano.name/ws/time.html#cron
8. Apple: Getting Started with launchd, URL: http://developer.apple.com/macosx/launchd.html
9. Microsoft: Task Scheduler (Windows), URL: http://msdn.microsoft.com/en-us/library/aa383614.aspx
10. Microsoft: At, URL: http://technet.microsoft.com/en-us/library/bb490866.aspx
11. Wikipedija: cron, URL: http://en.wikipedia.org/wiki/Cron
12. Wikipedija: at (Unix), URL: http://en.wikipedia.org/wiki/At_(Unix)
13. Wikipedija: launchd, URL: http://en.wikipedia.org/wiki/Launchd
14. Wikipedija: Task Scheduler, URL: http://en.wikipedia.org/wiki/Task_Scheduler
15. Wikipedije: at (Windows), URL: http://en.wikipedia.org/wiki/At_(Windows)

Dostupnost internetskih izvora provjerena je 14. lipnja 2010.

7. Naslov, sažetak i ključne riječi

Naslov

Sustav za odgođeno pokretanje poslova

Sažetak

Ovim su radom pokazane prednosti i nedostatci postojećih rješenja za vremenski uvjetovano pokretanje programa. Proučeni su postupci kao i poželjna svojstva suvremenog rješenja tog problema. Izrađeno je novo rješenje za GNU/Linux operacijski sustav koje demonstrira neke moderne značajke: programsko sučelje te ujedinjenje mogućnosti postojećih rješenja `cron` i `anacron` u jedan program.

Programsko sučelje osigurava proizvoljnim programima jednostavan i brz način upravljanja poslovima. Ujedinjenjem se značajki smanjuje uvišestručenje koda, pojednostavljuje se korištenje te se smanjuje ukupno opterećenje na sustav, a moguće je jer `anacron` samo proširuje mogućnosti `cron`a.

Ključne riječi

cron, anacron, at, GNU, Linux, raspoređivač poslova

8. Title, abstract and keywords

Title

Time based job scheduler

Abstract

This thesis showcases advantages and disadvantages of existing time based job schedulers. Desirable features of modern schedulers are studied. One such scheduler is programmed. It works under GNU/Linux operating system and comprises two modern features: it offers application programming interface and unifies capabilities of `cron` and `anacron`, two similar programs, into single program.

Application programming interface is major feature as it allows other programs to manage jobs in a simple and efficient manner. Unification of capabilities is possible due to the fact that `anacron` just extends capabilities of a `cron`. Benefits of it include lower code duplication, simpler usage and reduced load on a system.

Keywords

cron, anacron, at, GNU, Linux, job scheduler

Dodatak: Korištenje GNU razvojnog sustava

Na sljedećem će se jednostavnom primjeru pokazati korištenje GNU razvojnog sustava. Riječ je o klasičnom „Pozdrav, svijete!" primjeru. Datoteke i kazala koji će se stvarati nalaze se u kazalu `pozdrav/`.

Najprije se unutar korijenskog kazala projekta naprave dodatna kazala:

```
$ mkdir doc m4 src
```

Potom se stvore konfiguracijska datoteka `configure.ac` te predložak `Makefile.am`. Sadržaj datoteke `configure.ac` je:

```
$ cat configure.ac
AC_INIT([pozdrav],[0.0.1],[Ivan Horvat ivan@horvat.org],[pozdrav])
AM_INIT_AUTOMAKE(pozdrav,0.1)
AM_CONFIG_HEADER(config.h)
AC_PROG_CC
AC_PROG_INSTALL
AC_PROG_RANLIB

AC_CONFIG_FILES([
   Makefile
   src/Makefile
])
AC_OUTPUT
```

Naredba `AC_INIT` inicijalizira `configure` skriptu. Argumenti su ime paketa, broje inačice paketa, ime i adresa e-pošte autora te ime *tar* arhive (koristi se prilikom pakiranja paketa). Naredba `AM_INIT_AUTOMAKE` radi daljnju inicijalizaciju vezanu uz program `automake`; argumenti su ime paketa te broj inačice. Naredbom `AM_CONFIG_HEADER` zahtijeva se korištenje konfiguracijskog zaglavlja u kojem se mogu definirati razne konstante koje se koriste na raznim mjestima unutar projekta. `AC_PROG_CC` provjerava koji se prevodilac za programski jezik C koristi. `AC_PROG_INSTALL` provjerava postoji li na sustavu alat `install` – ako ne postoji, koristit će se skripta *install-sh*. Naredbu `AC_PROG_RANLIB` je potrebno staviti ako se stvara statička knjižnica. Argumenti naredbe `AC_CONFIG_FILES` su datoteke koje se stvaraju iz predložaka. Naredba `AC_OUTPUT` označava da `configure` skripta treba stvoriti datoteke predane kao argumenti naredbi `AC_CONFIG_FILES` iz pripadnih predložaka.

Datoteka `Makefile.am` sadrži sljedeće:

```
$ cat Makefile.am
EXTRA_DIST = configure
SUBDIRS = m4 doc src
```

Prvi redak datoteke označava da je datoteka `configure` dio distribucije, tj. mora biti prisutna u krajnjoj arhivi. Drugi redak označava da se ostatak distribucije nalazi u kazalima `m4/`, `doc/` i `src/`. Program `make` će se rekurzivno pozivati u tim kazalima, redom kojim su kazala navedena iza ključne riječi `SUBDIRS`; zbog toga i u tim kazalima moraju postojati predlošci `Makefile` datoteka. Zasada mogu biti prazne:

```
$ touch m4/Makefile.in doc/Makefile.in src/Makefile.in
```

Kako je spomenuto u poglavlju 3.1, GNU razvojni sustav zahtijeva postojanje nekih datoteka pa ih treba napraviti:

```
$ touch NEWS README AUTHORS ChangeLog
```

Zatim je potrebno konfigurirati paket. S obzirom na to da se GNU razvojni sustav sastoji od više alata, potrebno je pokrenuti više naredbi:

```
$ aclocal
$ autoconf
$ autoheader
$ automake -a
automake: configure.ac: installing `./install-sh'
automake: configure.ac: installing `./mkinstalldirs'
automake: configure.ac: installing `./missing'
automake: configure.ac: installing `./config.guess'
automake: configure.ac: installing `./config.sub'
```

Budući da se te naredbe trebaju pokrenuti nakon svake promjene datoteke `configure.ac` ili nekog od `Makefile` predložaka, praktično je te naredbe staviti u skriptu (obično naziva `reconf`). Nakon pokretanja tih naredbi, projekt se može izgraditi na standardni način, no pošto još ne sadrži nikakav programski kôd, to nema smisla.

Nakon prethodnih koraka može se pisati programski kôd. U ovom jednostavnom primjeru, kôd će se sastojati od samo jedne datoteke. Unutar kazala `src/` se napravi datoteka `zdravo.c` sljedećeg sadržaja:

```
$ cat src/zdravo.c
#include <stdio.h>

int main()
{
    printf("Pozdrav, svijete!\n");
    return 0;
}
```

Potrebno je izmijeniti predložak `src/Makefile.am` kako bi `automake` napravio ispravnu `Makefile` datoteku:

```
$ cat src/Makefile.am
bin_PROGRAMS = pozdrav
pozdrav_SOURCES = zdravo.c
```

Iza ključne riječi `bin_PROGRAMS` slijedi popis izvršnih datoteka (programa) koji će se izgraditi i instalirati. Nakon toga, potrebno je za svaki program definirati skup datoteka koje čine izvorni kôd tog programa. Ključne su riječi oblika `ime_programa_SOURCES`.

Pošto je mijenjana datoteka `src/Makefile.am`, potrebno je ponovno konfigurirati projekt (ukoliko je napravljena skripta `reconf`, dovoljno je nju pokrenuti). Konačno, projekt se može konfigurirati i izgraditi:

```
$ ./configure
(ispis maknut radi preglednosti)
$ make
```

Program se nalazi u kazalu `src/` te se može pokrenuti:

```
$ ./src/pozdrav
Pozdrav, svijete!
```

GNU razvojni sustav ima dodatnu mogućnost stvaranja distribucije u arhivi. Riječ je o distribuciji izvornog koda, a ne prevedenih (binarnih) programa. Kako bi se napravila distribucija u *tar* arhivi komprimiranoj *gzip* algoritmom, dovoljno je napraviti sljedeće:

```
$ make dist
```

Time se u korijenskom kazalu projekta stvara datoteka `pozdrav.tar.gz`.