Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Izrada Web i samostojeće aplikacije za elektromagnetski proračun sinkronih strojeva sa istaknutim polovima za područje srednjih i malih snaga

Autor: **Dinko Korunić, 0036355514**
Mentor: **Doc. dr. sc. Vlado Sruk**
Akademska godina: **2002/2003.**

Sažetak rada: Opisuje se specifičan dizajn Web i samostojeće aplikacije za proračun dotičnih strojeva sa naglaskom na prenosivosti, sigurnosti, pouzdanosti i pravovremenom otkrivanju grešaka. Posebice, riječ je o korištenju sinkronog I/O multipleksiranja za komunikaciju između višedretvenog poslužitelja i konkurentnih Web aplikacija. Osim toga, pojašnjena je izrada sigurnog i pouzdanog Unix daemona, kontrola signala i resursa u istom kao i izrada praktičnog Web sučelja prema takvom poslužitelju.

Ključne riječi: Unix daemoniziranje, sinkrono multipleksiranje, prenosivost, višedretvenost, kontrola resursa, kontrola signala, pouzdanost, detekcija pogrešaka, poslužitelj

Zagreb, 14. rujan 2003.

0. Sadržaj

0. Sadržaj 2
1. Uvod u zadanu problematiku 3
2. Opis programskog rješenja 4
  2.1 Sučelje za samostojeću aplikaciju 4
  2.2 Web sučelje 5
  2.3 Poslužiteljski dio aplikacije 7
  2.4 Proračunski dio aplikacije 8
3. Specifični primjeri programskog kôda 9
  3.1 Baratanje cookie podacima unutar Perla 9
  3.2 Baratanje CSV datotekama iz Perla 9
  3.3 Komunikacija s DBMS-om iz Perla 10
  3.4 Komunikacija Web s poslužiteljskom aplikacijom 11
  3.5 Komunikacija poslužiteljske sa Web aplikacijom 11
  3.6 Unix daemoniziranje 12
  3.7 Konkurentne dretve 14
  3.8 Kontrola resursa 15
  3.9 Kontrola signala 16
  3.10 Monitoring direktorija 16
  3.11 Respawnable proces 17
4. Opisi proračunskih algoritama 18
5. Zaključak 20
6. Literatura 22

1. Uvod u zadanu problematiku

Tema ovog seminara je uvod u problematiku i razrada rješenja aplikacije koja će biti napisana za Zavod ESA, FER. Specifično, riječ je o programu koji će obavljati elektromagnetski proračun sinkronih strojeva s istaknutim polovima za područje srednjih i malih snaga. Postoji potreba za takvom aplikacijom koja će omogućiti studentima da tijekom laboratorijskih vježbi mogu provjeriti ručno dobivene specifikacije. Sam proračun je relativno jednostavan matematički postupak pomoću kojeg se odrede glavne dimenzije generatora, projektira se namot, napravi se proračun magnetskog kruga, dimenzionira uzbudni namot i napokon se proračunaju gubici.
Tema je opisivanje programerski zanimljivog materijala, budući da su numerički postupci potrebni za navedeni proračun poznati i ispitani, te dostupni iz odgovarajuće literature [13][14]. Sama aplikacija je zamišljena takvom da će biti postavljena na Web stranice Zavoda ESA (*http://esa.fer.hr/*) i osim toga da će biti dostupna u samostojećim (*desktop*) inačicama za Unix i Win32 platforme. Jasno, ona sama će biti pod GNU licencom kao posve otvoren kôd (no, samo poslužiteljski dio i oba sučelja, budući da su algoritmi "zaštićeni") što je poželjno za aplikacije proizašle iz akademske zajednice, budući da doprinose zajednici ne samo svojim radom već i kôdom koji je moguće slobodno nadopunjavati i ona sama može predstavljati početak za nečiju nadogradnju. No, ove specifikacije nam nameću određene posebnosti u dizajnu i arhitekturi aplikacije i samim time čine ovaj problem zanimljivijim, pa ćemo o tome detaljnije u idućem poglavlju.
Najvažnija poglavlja ovdje se bave sa problematikom koja je većini Unix programera poznata: kako učiniti određeni servis sigurnim (odbaciti administratorske privilegije, zaštititi ga od svih forma prepisivanja sistemskog stoga), kako ga učiniti pouzdanim (oporavak od pogreške, kontrola signala i resursa), kako riješiti konkurentnost (višedretvenost, zaključavanje, itd.) te kako riješiti komunikaciju sa klijentima. Ideja koja se iznosi u ovom radu slijedeća: Stvoriti dijeljeni direktorij u kojem se nalaze određene datoteke za zahtjevima. Zahtjevi su jednostavne tekstualne CSV (Comma Separated Values) datoteke koje stvaraju Web CGI (Common Gateway Interface) programi [7]. Ime svakog tog zahtjeva odnosno datoteke se stvara jedinstveno - od vremena zahtjeva i jedinstvenog broja procesa same pojedinačne Web aplikacije. Nadalje, taj dijeljeni direktorij "osluškuje" sama proračunska aplikacija u osnovnoj dretvi svakih nekoliko sekundi. Po prispjelom zahtjevu ona, u slučaju da predodređeni broj dretvi još nije popunjen, stvara novu dretvu koja kao argument dobiva dotičnu datoteku. Dotična dretva obavlja svoj posao proračuna i vraća nazad dobivene podatke na koje svaka individualna Web aplikacija čeka u "usnulom" stanju (dotično se razrješava sistemskim I/O multipleksiranjem). Dretva nakon obavljenog posla prekida sa radom i "umire", a Web aplikacija prikaže podatke na zaslonu zajedno sa grafovima nacrtanim iz dobivenih CSV tablica. Sam poslužitelj je tako dizajniran da zauzima minimum sistemskih resursa, te da su vremena osluškivanja direktorija i broj dretvi kao i priručni podaci za računanje (datoteke sa karakteristikama materijala, itd.) sasvim izmjenjivi. Osim toga, ovakva aplikacija nužno mora biti potpuno prenosiva [1][3] i funkcionalna na većini dostupnih platformi - dakle Unixoidima i Windows baziranim operativnim sustavima.

2. Opis programskog rješenja

Ponovimo ukratko željene posebnosti ovog programa:

- multiplatformnost i prenosivost,
- otvoren kôd,
- korištenje postojećih standardnih biblioteka,
- mogućnost izrade samostojeće i poslužiteljske aplikacije.

Iz ovih zahtjeva proizlazi potreba da samu aplikaciju arhitekturalno razdijelimo na ove programske dijelove:

- CLI (CommandLine Interface) i GUI (Graphical User Interface) sučelje,
- Web sučelje,
- poslužiteljski dio aplikacije,
- CLI programski dio za proračun.

Možemo vidjeti da su ovdje zadovoljeni principi modularnosti - sučelje je sasvim izmjenjivo i nezavisno od glavnog proračunskog dijela, dok je serverski dio aplikacije samo nadogradnja na CLI proračunski dio i predstavlja dio koji će odgovarajućim preprocesorskim naredbama biti ugašen za krajnje korisnike samostojeće aplikacije. Ne samo to, već smo ovakvom nezavisnošću postigli i maksimalnu prenosivost programa koji će u različitim okruženjima i platformama moći raditi neovisno. No, pojasnimo detalje o svakoj komponenti...

2.1 Sučelje za samostojeću aplikaciju

Kao što smo već naveli, ovo sučelje za samostojeću aplikaciju treba biti nadogradnja na isključivo računski (CLI) dio i treba omogućiti da je on apsolutno prenosiv i nezavisan. Nadalje, ideja je da će dotično sučelje skupiti odgovarajuće parametre iz interakcije s korisnikom (popunjavanje polja u nekoj formi) i/ili po potrebi dohvatiti podatke iz lokalne baze podataka koja može biti u proizvoljnoj formi (MySQL, CSV delimitirana datoteka, PostgreSQL, itd.). Poseban naglasak ćemo staviti na CSV datoteku budući da će ona omogućiti studentima i asistentima pregled dosadašnjih proračuna i iz Microsoft Excela i sličnih programa, kao i jednostavno editiranje u drugim alatima.
Postavlja se i problem izrade sučelja. Ona bi bila platformski posve ovisna i koristila bi različite biblioteke: dok je kod Windowsa riječ o GUI rješenju korištenjem WinAPI-ja i pripadnih biblioteka Windowsa, pod Unixom i Unixoidima imamo mogućnost korištenja ne samo GUI X11 biblioteka već i CLI orijentirane Ncurses biblioteke. No, moguće je alternativno rješenje - koristiti Cygwin projekt koji će omogućiti korištenje Unixoidnih biblioteka (i STL nadogradnji iz libstdc++ za proračunski dio, naravno) i samim time ćemo imati jedno jedinstveno i potpuno portabilno sučelje. Jasno, pod Unixoidima ćemo koristiti GCC za dobivanje izvršnog koda, dok pod Win32 arhitekturom odgovarajuće Cygwin okruženje u vidu Win32 inačica GCC-a i Mingw32 (Win32 *cross compiler*). Time ćemo moći zadržati jedinstveni kôd koji će raditi nesmetano na obje platforme. Dakle, da sumiramo karakteristike sučelja:

1) koristiti će Ncurses (CLI) i/ili Xlib (GUI) biblioteke,
2) moći će čitati i pisati lokalnu CSV datoteku s dosadašnjim proračunima,
3) imat će pregled dosadašnjih proračuna,
4) po završetku proračuna će ispisati podatke u preglednom obliku: omogućiti će listanje po stranicama itd,

5) imati će osnovne provjere za pogreškama:
   - postoji li CSV datoteka i da li je moguće čitati i pisati - ovisno o dozvolama će i omogućiti samo pregled ili i upis,
   - postoji li program za proračun,
   - postoji li dovoljno resursa za proračun (dodavanje u CSV datoteku, itd.),
   - jesu li u polja upisane "krive" vrijednosti - jasno, to je će biti trivijalno provjeravanje 0 u vrijednostima budući da će detaljnije provjere imati proračunski dio,
   - omogućiti će obradu izlaznih kôdova proračunskog programa i prijavu pogreške korisniku.
6) kontrolirat će osnovne signale (KILL, INT, TERM) i omogućiti izlaženje iz programa na prirodan način,
7) detektirat će sve karakteristike materijala u vanjskim datotekama i omogućavati njihovo korištenje - dakle biti će moguće obavljati proračun sa bilo kakvim materijalom uz one koji će biti digitalizirani (koji će standardno dolaziti uz aplikaciju).

2.2 Web sučelje

Primarna zadaća cijele ove aplikacije sa svim komponentama je zamišljena za opsluživanje studenata preko Weba. Jasno, tu se pojavljuje nekoliko osnovnih problema kao i kod svakog servisa:

- Konkurentnost: Studenti će paralelno pristupati vježbama i izvršavati ih neovisno jedan o drugome, dakle treba riješiti i zaključavanje i ostale probleme paralelizma.
- Potrošnja resursa: Hoćemo li dozvoliti da svaki student ima vlastitu inačicu proračuna koja je memorijski i procesorski zahtjevna i predstavlja potrošača?
- Sigurnost: Omogućiti li direktnu komunikaciju između Web preglednika preko Web poslužitelja u samu aplikaciju, znajući da to predstavlja sigurnosni propust? Kako promijeniti korisnika i grupu u međuvremenu na elegantan i siguran način? kako odbaciti potrebne dozvole za *cgi-bin* program? Ostaviti li proces u vlasništvu korisnika "*nobody*"? [11]
- Modularnost: Kako omogućiti dotičnom sučelju da može pristupati i MySQL i PostgreSQL i CSV datotekama na što jednostavniji i potencijalno izmjenjiv način?
- Portabilnost: Kako napisati Web aplikaciju koja će raditi i u Apacheu i pod Microsoft IIS-om?

Kao prirodan odgovor na ovakve probleme nudimo arhitekturalno rješenje koje je već implicitno ponuđeno u općem uvodnom dijelu: odijeliti proračunski dio od dijela koji služi samo kao sučelje. I ne samo to, nego ćemo čak i promijeniti jezik u kojem će biti Web sučelje i napisati ga u jednom od najmoćnijih interpretiranih jezika - Perlu (Larry Wall's Practical Extraction and Report Language) [2][6][8][9][10][12]. Navedimo nekoliko razloga za takav odabir:

- CPAN - moćna baza već gotovih modula, koja će nam omogućiti nekoliko važnijih modula već pripremljenih za naše potrebe:
  - Perl DBI (Database Interface) modul i specifično DBD::CSV, DBD::File kao upravljački programi za pristupanje individualnim datotekama, te DBIx::Recordset za pristupanje različitim DBMS-ovima - koji nam trebaju omogućiti da korisnik može unijeti vlastiti završeni proračun ili na samom početku vidjeti neki tuđi proračun sa sličnim podacima. Dotični moduli nam daju mogućnost pretraživanja, dohvata i pohrane željenih podataka

pomoću SQL naredbi.
    - CGI::Cookie modul koji će nam omogućiti da korisnik ima lokalno spremljene podatke koje je zadnji put upisao u formu, što će ne samo pomoći pri svakom idućem unosu, već i kao rješenje gubljenja unesenih podataka u slučaju korisnikovih problema s Web pretraživačem, vezom do Web poslužitelja, itd.
    - različiti Perl moduli za autentifikaciju koji će omogućiti administratoru da udaljeno održava listu spremljenih podataka (Apache Auth i sl.)
- Sigurnost, minimalni zahtjevi, segmentiranje zadataka: dizajn je zamišljen tako da Perl CGI skripta služi kao međunivo između proračunskog dijela i korisnika. Ovdje su same privilegije CGI skripte nikakve (ovisno o konfiguraciji Apache poslužitelja može biti u vlasništvu korisnika "*nobody*" ili korisnika "*www-data*" i sl.) i ona može samo poslati SQL orijentirani poziv DBMS-u (dohvati, pohrani, potraži) koji je standardno vremenski ograničen te predati proračunskom dijelu i čekati na odgovor. Zahtjev za resursima je tako minimalan, budući da će tek proračunski dio koji je zahtjevan za resursima moći biti konkurentan, a skripta čeka do završetka proračuna ili pogreške koristeći neki od *select*, *poll* i sličnih sistemskih događajno upravljanih poziva koji će omogućiti da program bude u S-stanju (da "spava") sve dok se ne desi kakva promjena na opisniku datoteke. Osim toga, Perl kao takav je iznimno prigodan za rudimentarne provjere unosa, budući da nema nikakvih problema sa prepisivanjem stoga. Veličine varijabli (jednostavnih varijabli, matrica, vektora, indeksiranih polja) kod njega su posve dinamički određene. Dapače, tipična je Unixoidna filozofija svaki problem razdijeliti na niz programa koji obavljaju točno po jedan zadatak - ali ga obavljaju dobro.
- Brzina izvođenja: postoji poseban mehanizam za Apache poslužiteljski softver koji se zove Perl::Registry unutar *mod_perl* ekstenzije i omogućava spremanje Perl metaobjekata (koji nastaju interpretiranjem Perl izvornog koda) u za to predodređenu priručnu memoriju. Jasno, ovaj mehanizam omogućava da se sva slijedeća izvršenja određenog koda (u ovom slučaju CGI skripte) bitno ubrzavaju jer je metaobjekt već pripremljen i nema potrebe za ponovnim interpretiranjem.

Sumirajmo dakle ponovno zadatke koji se postavljaju pred Web sučelje u formi Perl CGI skripte:

- raspolaganje (slanje i primanje) HTTP *cookie-jima*,
- prihvat podataka od korisnika,
- prihvat već izračunatih podataka od odgovarajućeg DBMS-a ili iz CSV datoteke (modularno i konfigurabilno),
- pohrana izračunatih podataka preko odgovarajućeg DBMS-a ili u CSV datoteku (modularno i konfigurabilno),
- čekanje na proračunski dio pomoću sistemskih poziva (detekcija uspješne/neuspješne predaje zahtjeva i detekcija uspješnog/neuspješnog preuzimanja proračuna),
- osnovne provjere za pogreškama (upisane nule u poljima, krivi *cookie*, itd.),
- funkcioniranje neovisno o operacijskom sustavu i Web poslužitelju,
- minimum potrebnih privilegija i resursa za rad.

2.3 Poslužiteljski dio aplikacije

Ovaj dio aplikacije je zamišljen kao nadogradnja samog proračunskog dijela i to kao rješenje slijedećih specifičnih problema:

- Konkurentnosti: Bit će pokretano nekoliko inačica ovog resursno zahtjevnog programa, stoga je potrebno napraviti mehanizam koji će omogućiti kontrolu konkurentnosti,
- Sigurnost: Aplikacija mora biti specifično dizajnirana kao standardni Unix *daemon* [4] i potrebno je strukturu programa i varijable predodrediti na takav način da niti jedna forma prepisivanja stoga nije moguća (ovime je potrebno obuhvatiti ne samo standardna prepisivanja varijabli, već i *string* format pogreške i ine sigurnosne probleme),
- Sistemski resursi i pouzdanost: Nužno je implementirati slijedeće karakteristike:
    - kontrolu resursa i same aplikacije,
    - minimizirati broj inicijalizacija,
    - sistemsko "spavanje" procesa u trenucima neaktivnosti,
    - ispravno čišćenje vlastitog memorijskog prostora prije izlaska programa,
    - vraćanje ispravnih izlaznih kodova operacijskom sustavu,
    - mogućnost nadzora aplikacije i provjere statusa,
    - kontrolu sistemskih signala i standardne reakcije na HUP, KILL, TERM signale,
    - mogućnost da proces bude uvijek aktivan u sustavu - dakle oporavak od pogreške.

Imajući navedene karakteristike u vidu formira se slijedeće rješenje:

- napraviti dio aplikacije u vidu Unix *daemona*:
    - rješava se svih alociranih terminalskih uređaja i zatvara sve dobivene opisnike datoteka,
    - ovisno o danim argumentima, program automatski odlazi u pozadinu, te odbacuje administratorske privilegije,
    - postavlja kontrole signala na vlastite odgovarajuće funkcije,
    - otvara opisnik datoteke i čeka na zahtjeve koristeći sistemske pozive tipa *poll* i *select* koji su događajno bazirani, odnosno proces je u S-stanju sve do trenutka promjene stanja opisnika datoteke,
    - pri svakom zahtjevu radi novu dretvu u kojoj vrši proračun i pri završenom proračunu vraća u datoteku odgovarajuće podatke,
    - datoteka je u svim kritičnim odsječcima zaštićena zaključavanjem sistemskim funkcijama.
- aplikacija je na opisani način zatvoreni sustav sa sučeljem, no potrebno je omogućiti i apsolutnu pouzdanost:
    - program se podešava kao nadgledani *daemon* od strane DJB *daemontoolsa* koji omogućavaju kontrolu servisa, provjeru statusa, automatsko restartanje, nadgledanje sistemskih logova itd,
- aplikacija je kompletno napisana u C++ jeziku koristeći STL ekstenzije čime se:
    - dobiva mogućnost korištenja "*string*", "*vector*" i ostalih tipova varijabli i kolekcija sa dinamički određenom veličinom,
    - omogućava korištenje naprednijih *iostream* i inih funkcija, izbjegavši sve nedostatke standardnih C inačica,
- sve potrebne postavke aplikacija može dobiti dvojako:
    - preko varijabli okruženja,

- preko argumenata aplikaciji.

Osim toga Web sučelje će detektirati sve karakteristike materijala u vanjskim datotekama i omogućavati njihovo korištenje - dakle biti će moguće će obavljati proračun s bilo kakvim materijalom uz one koji će biti digitalizirani (koji će standardno dolaziti uz aplikaciju). Osim toga, uz odgovarajuće argumente će "administratori" biti u mogućnosti dodavati lokalno izvedene CSV datoteke s digitaliziranim karakteristikama materijala.

2.4 Proračunski dio aplikacije

Elektromagnetski proračun sinkronog generatora može se podijeliti u nekoliko cjelina:

- proračun osnovnih veličina (faktor namota, Karterov faktor, zubi statora, jaram statora, polovi rotora, idealna duljina, karakteristične veličine),
- karakteristike praznog hoda (magnetski krug pri nazivnom naponu, magnetski krug za različite napone),
- proračun armaturnog namota (geometrije, električnog otpora),
- proračun uzbude za opterećeno stanje (uzbudno protjecanje, regulacijske karakteristike, itd.),
- proračun uzbudnog namota (izvedba i dimenzioniranje, namoti, itd.),
- proračun prigušnog namota (dimenzije, protjecanja, struja, otpori, masa),
- proračun reaktancija i vremenskih konstanti,
- proračun gubitaka i stupnja korisnosti (masa paketa statora, gubici praznog hoda, gubici kratkog spoja, gubici u uzbudnom namotu),
- proračun masa aktivnih materijala i mehaničke vremenske konstante (masa bakra, željeza, zamašna masa),
- proračun ostalih detalja kao dopuna osnovnom proračunu.

Tijekom proračuna za izradu paketa statora i polova se koriste podaci iz digitaliziranih karakteristika materijala. Ovisno o zadanim materijalima, čitaju se krivulje magnetiziranja i pomoću njih se proračunavaju tablice praznog hoda. Unošenje dodatnih materijala izvodi se preko Web sučelja ili pak ručnim dodavanjem CSV datoteka sa karakteristikama, pri čemu aplikacija sama prepoznaje datoteke s materijalima kao i samo sučelje koje ih onda nudi na odabir.

3. Specifični primjeri programskog kôda

Kompleksnost ove aplikacije nalaže detaljnu razradu i analizu programskih odsječaka specifičnih za pojedinu problematiku. Zbog opsežnosti koda, slijede samo pojedinačni zanimljivi i kompleksniji slučajevi:

3.1 Baratanje cookie podacima unutar Perla

Slijedeći primjer pokazuje stvaranje *cookie* objekta, prihvaćanje i odašiljanje korisnikovom Web pregledniku korištenjem CGI::Cookie modula. Upravo ovakav odsječak nalazit će se u Web aplikaciji i omogućit će da korisnikov vlastiti Web preglednik "pamti" zadnje postavke i unesene vrijednosti:

```
use CGI qw/:standard/;
use CGI::Cookie;

# stvaramo novi kolacic s podacima koje korisnik napuni
# u formi, stavimo istek na 1 mjesec i posaljemo
cookie = new CGI::Cookie(
  -name =>  'esa-proracun',
  -value   =>  ['vrijednost1', 'vrijednost2', 'vrijednost3']
  -expires =>  '+1M',
  -domain  =>  '.esa.fer.hr',
  -path    =>  '/cgi-bin/esa-proracun'
  -secure  =>  0);

# postavimo kolacic (CGI.pm)
print header(-cookie=>[$cookie1,$cookie2]);

# postavimo kolacic kao string (bez CGI.pm)
print "Set-Cookie: ",$c->as_string,"\n";

# prihvatimo postojece kolacice
%cookies = fetch CGI::Cookie;

# uzmemo podatke iz naseg kolacicica iz kolekcije
# svih kolacica
$data = $cookies{'esa-proracun'}->value;
```

3.2 Baratanje CSV datotekama iz Perla

Ovaj odjeljak koda odnosi se na mogućnost Perla da korištenjem o DBI modula i odgovarajućeg upravljačkih programa (specifično DBD::CSV) pristupa datotekama i upravlja njihovim sadržajem standardnim SQL upitima:

```
use DBI;

# spojimo se i stvorimo praznu CVS bazu
$dbh = DBI->connect("DBI:CSV:f_dir=/var/www/esa-
proracun/materijali")
```

```
  or die "Ne mogu se spojiti: " . $DBI::errstr;
$sth = $dbh->prepare("CREATE TABLE materijal (id INTEGER,
name CHAR(10))")
  or die "Ne mogu pripremiti: " . $dbh->errstr();
$sth->execute()
  or die "Cannot execute: " . $sth->errstr();
$sth->finish();
$dbh->disconnect();

# prihvatimo iz CSV datoteke kompatibilne sa Excelom
# podatke odijeljene sa ';'
$dbh = DBI->connect("DBI:CSV:f_dir=/etc;
  csv_sep_char=\\;")
$dbh->{'csv_tables'}->{'materijal'} = {
  'col_names' => ["b-karakteristika", "h-karakteristika"]};
$sth = $dbh->prepare("SELECT * FROM materijal WHERE b-
karakteristika > 0.8 ORDER BY h-karakteristika");
```

3.3 Komunikacija s DBMS-om iz Perla

Slijedi hipotetski primjer komunikacije s postojećim bazama podataka, u ovom slučaju MySQL poslužiteljem (koristeći DBI i DBD::mysql i DBD::mSQL) poradi dohvaćanja spremljenih dosadašnjih proračuna ili pak digitaliziranih karakteristika materijala:

```
use DBI();

# spajamo se na MySQL bazu
$dbh = DBI->connect("DBI:mysql:database=proracun;
  host=localhost", "proracun", "proracunlozinka",
  {'RaiseError' => 1});

# ispraznimo tablicu proracuna (npr. administratorska
mogucnost)
eval { $dbh->do("DROP TABLE proracun") };
print "Brisanje tablice nije uspjelo: $@\n" if $@;

# stvaramo novu tablicu
$dbh->do("CREATE TABLE proracun (id INTEGER, name
VARCHAR(20))");

# sad ubacimo podatke u tablicu
$dbh->do("INSERT INTO proracun VALUES (1, " .
  $dbh->quote("Materijal1") . ")");
# i sad dohvatimo te podatke
$sth = $dbh->prepare("SELECT * FROM proracun");
$sth->execute();
while (my $ref = $sth->fetchrow_hashref())
{
  print "Redak materijala: id = $ref->{'id'},
    ime = $ref->{'name'}\n";
}
```

```
$sth->finish();

# odspoji se od baze
$dbh->disconnect();
```

3.4 Komunikacija Web s poslužiteljskom aplikacijom

Vjerojatno jedan od složenijih problema u komunikaciji je problem sinkronizacije dvaju ili više programa koji čitaju jednu datoteku (preciznije, jedan opisnik datoteke). Srećom, na Unix platformama postoji nekoliko sistemskih poziva i mehanizama koji to omogućavaju - *poll()*, *select()*, *epoll()*, *kqueue()* i */dev/poll* [5]. Ova prva dva su karakteristična za većinu standardnih Unixoida, dok je treći moguće naći tek u modernim Linux 2.6 *kernelima*. Četvrti se, nažalost, nalazi samo u FreeBSD-ju, a peti na Solaris operativnom sustavu i na posebno *patchiranom* Linuxu. Specifično, riječ je o mehanizmima koji omogućavaju izvršavanje predodređenih funkcija kada se odigra kakav specifičan događaj na nekakvoj datoteci (npr. FIFO datoteci) ili kada prođe određena količina vremena. Opisana aplikacija će koristiti isključivo *poll/select* par upravo zbog najveće rasprostranjenosti u operativnim sustavima. Također, očito je da oni predstavljaju rješenje komunikacije kad jedan program (Web aplikacija) predaje drugom programu (proračunskom dijelu) određene podatke preko datoteka i čeka u S-stanju na promjenu događaja u datoteci. Naravno, *select* poziv u Perlu je izveden pozivanjem ekvivalentnog *select()* C-olikog sistemskog poziva koji predstavlja sinkrono I/O multipleksiranje:

```
# primjer Perl sucelja:

# otvorimo r/w datoteku zahtjev.PID
open FD, "+>", zahtjev.$$;

# zapisemo nas zahtjev
syswrite FD, $msg;

# cekamo na odgovor i procitamo ga
$rin = '';
vec($rin, fileno(FD), 1) = 1;
select $ready = $rin, undef, undef, $timeout;
if (vec($ready, fileno(FD), 1) == 1)
{ $bytes = sysread FD, $mesg; }
close FD;
```

3.5 Komunikacija poslužiteljske sa Web aplikacijom

Slijedi i primjer *select()* odjeljka u C/C++ jeziku koji će upravo na predočen način biti implementiran u poslužiteljskoj inačici (datoteku se može napraviti ručno sa "*mknod datoteka p*" pri čemu se stvara FIFO datoteka):

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/select.h>

int main()
{
  fd_set rfds;
  struct timeval tv;
  int retval, fd;

  fd = open("/datoteka/za/komunikaciju",
    O_SYNC | O_RDWR);
  if (fd == -1)
  {
    perror("Ne mogu otvoriti datoteku");
    exit(EXIT_FAILURE);
  }

  /* iscekujemo promjenu na fd, pa ga dodajemo u set */
  FD_ZERO(&rfds);
  FD_SET(fd, &rfds);

  /* cekat cemo tocno 60 sekundi */
  tv.tv_sec = 60;
  tv.tv_usec = 0;

  /* udji u cekanje */
  retval = select(fd + 1, &rfds, NULL, NULL, &tv);

  if (retval == -1)
    perror("Greska u select() pozivu");
  else
  if (retval)
   puts("Podaci su citljivi i dostupni");
   /* FD_ISSET(0, &rfds) je istinit */
  else
    fputs("Vrijeme cekanja je isteklo, "
      "podaci nisu bili dostupni", stderr);

  return EXIT_SUCCESS;
}
```

3.6 Unix daemoniziranje

Pojednostavljeno rečeno, Unix *daemone* možemo promatrati kao programe koji se izvršavaju u pozadini odnosno neinteraktivno (nezavisno od ikakvog korisničkog terminala) i čine neku funkciju na sustavu. *Daemon* je kratica za Disk And Execution MONitor, i danas postoje tisuće različitih Unixoidnih i inih takvih procesa. Mogućnost da operativni sustav izvršava nekakav *daemon* znači da će na sustavu najvjerojatnije biti niz aplikacija koje se paralelno izvršavaju: uz korektno napisani kod takve aplikacije najčešće

se izvršavaju beskonačno dugo (odnosno do pada ili gašenja cijelog sustava). Jedan od najvažnijih načina kako se poruke o stanju *daemona* i pogreškama dojavljuju sustavu su tzv. log servisi, kojima se obraća daemon posebnim sistemskim pozivima.
Procedura za stvaranje ispravnog *daemona* je nešto složenija, stoga slijedi i pseudokod za preporučeno stvaranje [4]:

- šalje se program u pozadinu:
  - stvoriti novo dijete-proces pomoću *fork()*,
  - prekinuti rad procesa roditelja s *exit()*.
- ponovno se šalje u pozadinu nakon prekidanja veze sa terminalom (ovo obično nije potrebno, ali predstavlja korektno izveden daemon):
  - stvara novi session sa *setsid()*,
  - ignorira se signal HUP,
  - stvara se novo dijete-proces,
  - prekida se rad procesa roditelja.
- ponovno se odjeljuje proces s *fork()* i *exit()* da bi se razriješio mogući problem s kontrolom terminala,
- mijenja se radni direktorij,
- postavlja se bitovna maska za stvaranje novih datoteka (*umask*),
- zatvaraju se svi otvoreni opisnici datoteka (obično prvih 64),
- otvara se komunikacija prema sistemskim logovima sa *openlog()*.

Uz ove standardne korake, često se nalazi i dodatan sigurnosni model:

- postavljaju se dodatne korisničke grupe s *initgroups()*,
- mijenja se pokazivač osnovnog datotečnog sustava u procesu s *chroot()*,
- mijenja se radni direktorij u direktorij unutar *chroot* okoline,
- otpuštaju se administratorske privilegije i grupe sa *setgid()* i *setuid()*.

Specifično, navedimo potpuno korektan primjer s pravilnim rukovanjem potencijalnim pogreškama:

```
/* saljemo proces u pozadinu */
if ((pid = fork()) == -1)
{
  perror("neuspjeli fork");
  exit(EXIT_FAILURE);
}
else
  if (pid > 0)
  /* parent exits */
  exit(EXIT_SUCCESS);

/* novi session leader */
if ((pid = setsid()) == -1)
{
  perror("setsid");
  exit(EXIT_FAILURE);
}

/* promijenimo radni direktorij */
if (chdir("/radni/direktorij") < 0)
```

```
{
  perror("neuspjeli chdir");
  exit(EXIT_FAILURE);
}

/* nova maska - obicno 022 */
umask(nas_umask);

/* zatvorimo opisnike */
for (i = 0; i < 64; ++i)
  close(i);

/* inicijaliziramo sistemsko logiranje */
openlog("imeprograma", LOG_ODELAY, LOG_DAEMON);

/* preuzmemo dodatne grupe */
if (initgroups (info->pw_name, info->pw_gid) == -1)
{
  perror("neuspjeli initgroups");
  exit(EXIT_FAILURE);
}

/* mijenjamo procesni root pokazivac */
if (chroot("/radni/direktorij") < 0)
{
  perror("neuspjeli chroot");
  exit(EXIT_FAILURE);
}

/* promijenimo GID */
if (setgid(neki_gid) < 0)
{
  perror("neuspjeli setgid");
  exit(EXIT_FAILURE);
}

/* promijenimo UID */
if (setuid(neki_uid) < 0)
{
  perror("neuspjeli setuid");
  exit(EXIT_FAILURE);
}

/* posaljemo sistemsku poruku.. */
syslog(LOG_INFO, "nas daemon je uspjesno podignut");

/* i tako dalje */
```

3.7 Konkurentne dretve

Slijedi i vrlo pojednostavljen primjer stvaranja polja samostalnih (*detached*) dretvi - kao

što će i proračunski dio raditi. U ovom slučaju svaka dretva posve neovisno o početnom programu obavi svoj posao i preda rezultat u pripadajuću FIFO datoteku. Također, kao što je već opisano, osnovna dretva stvara točno određeni unaprijed definirani broj dretvi:

```
/* vektor sa svim TID-ovima svih dretvi */
pthread_t tid[maksimalno_dretvi];

/* postavimo karakteristike samostalne dretve */
pthread_attr_t detach_attr;

pthread_attr_init(&detach_attr);
pthread_attr_setdetachstate(&detach_attr,
  PTHREAD_CREATE_DETACHED);

/* stvorimo jednu individualnu dretvu*/
pthread_create(tid[rednibroj], &detached_attr,
  funkcija_proracuna, dobiveni_datotecni_deskriptor);

/* itd. */
```

3.8 Kontrola resursa

Jedan od vrlo važnih detalja je i mogućnost prepoznavanja kada je u procesu nešto krenulo krivo, odnosno kada je proces zbog programerske ili neke druge pogreške krenuo preuzimati velike količine memorije. Ovakvi slučajevi se ne bi smjeli dešavati, no poradi pouzdanosti rada na višekorisničkom poslužitelju poželjno je imati kod koji će ugasiti proces u slučaju prevelike potrošnje sistemskih resursa, recimo radne memorije. Pod Unixom se to postiže korištenjem *setrlimit()* poziva na RLIMIT_DATA (veličina podatkovnog segmenta: inicijalizirani podaci, neinicijalizirani podaci i stog) i RLIMIT_AS (ukupna količina memorije koju proces može koristiti). Prvi utiče na *srbk()* i *brk()* pozive, dok drugi utiče i na *mmap()* i ostale derivate (*calloc()*, *alloca()* i *malloc()*):

```
char *str;
struct rlimit myrlimit;

/* postavimo limite sistemskih resursa */
getrlimit(RLIMIT_DATA, &myrlimit);
myrlimit.rlim_cur = 10000000;
setrlimit(RLIMIT_AS, &myrlimit);

/* pokusamo alocirati nedozvoljenu vrijednost */
str = (char *) malloc(11000000);

/* i program ce automatski izaci */
if (str == NULL)
{
  perror("malloc nije uspio");
  exit(EXIT_FAILURE)
}
```

3.9 Kontrola signala

Rukovanje sistemskim signalima je relativno jednostavno implementirati i u ovoj aplikaciji će isključivo služiti za potpunost *daemona*. Dakle, potrebno je hvatati signale HUP (učitavanje ponovo biblioteke materijala), TERM i INT (izlazak iz programa), USR1 (ispisivanje statistike preko *syslog()*). Teoretski je moguće hvatati i SEGV i vratiti spremljeno stanje stoga prije pogreške, no to se obično ne preporuča. Kombiniranje signala sa dretvama je nešto složeniji proces, no ovdje je pojednostavljeni primjer:

```
void moj_hup(int n)
{
  /* ucitaj ponovo materijale, itd.. */
}

void moj_izlazak(int n)
{
  _exit(EXIT_SUCCESS);
}

void moja_statistika(int n)
{
  /* ispisi statistike, itd. */
  syslog(...);
}

signal(SIGHUP, moj_hup);
signal(SIGTERM, moj_izlazak);
signal(SIGTERM, moj_izlazak);
signal(SIGUSR1, moja_statistika);
```

3.10 Monitoring direktorija

Web aplikacije (naravno, ima ih više budući da se posluživanje odvija prema cijelom nizu klijenata) moraju na siguran i jednostavan način predavati svoje zahtjeve poslužitelju. Da bi se to odvijalo uspješno, postoje dvije mogućnosti:

- predavati zahtjeve preko kakve specijalne datoteke (npr. *Unix domain socket*),
- predavati zahtjeve preko zajedničkog direktorija.

Ovo posljednje rješenje je nešto kompliciranije ali i prenosivije. Stoga slijedi izvadak relevantnog izvornog kôda:

```
void pokreni(char *ime)
{
  struct stat st;

  /* rijec je o ".." ili ".", stoga nastavi dalje */
  if (fn[0] == '.')
    return;

  /* saznaj podatke o datoteci */
  if (stat(fn, &st) == -1)
```

```
  {
    perror("Ne mogu provjeriti datoteku");
    return;
  }

  /* provjeri da li je FIFO */
  if ((st.st_mode & S_IFMT) != S_FIFO)
    return;

  /* dalje kod barata s FIFO datotekom */
  /* kao sto je prikazano u prethodnim primjerima */
}

void pregledaj(void)
{
  DIR *dir;
  direntry *d;

  /* pristupi direktoriju */
  dir = opendir(".");
  if (!dir)
  {
    perror("Ne mogu citati direktorij");
    exit(EXIT_FAILURE);
  }

  /* obradjuje sve datoteke u direktoriju slijedno */
  for (;;)
  {
    errno = 0;
    d = readdir(dir);
    if (!d)
      break;
    pokreni(d->d_name);
  }

  perror("Ne mogu citati direktorij");
  closedir(dir);
  return(EXIT_FAILURE);
}
```

3.11 Respawnable proces

Naposljetku, treba spomenuti i problem "gašenja" *daemona* - neželjenog prestanka rada uzrokovanog internom pogreškom ili DoS (Denial of Service) napadom. Proces je moguće označiti "*respawnable*" zastavicom dodavanjem u sistemsku *inittab* konfiguraciju. Alternativna mogućnost je postavljanje nadzornog programa (tipa *daemontools* ili *monit*) u inittab i njegovo konfiguriranje da prati servisni proces. Ovakvim postavkama proces nikad ne umire, odnosno čak i kad se ugasi, *kernel* se brine da se proces ponovno automatski pokreće i tako u beskonačnost. Naravno, potrebno je pobrinuti se da takav proces nikad ne zauzima nepotrebno resurse (memoriju, CPU, datotečne opisnike, itd.).

4. Opisi proračunskih algoritama

Proračunski dio aplikacije prolazi kroz točno određene faze. Redom, to su ovi algoritmi za proračun sinkronog stroja:

1) proračun osnovnih veličina:
   - faktor namota (zonski, tetivni, uslijed skošenja, ukupni),
   - Karterov faktor (statora, rotora, ukupni),
   - zubi statora: ovalni utor, ovalno-trapezni utor, poluotvoreni pravokutni utor, otvoreni pravokutni utor (širina zuba, površina presjeka, magnetsko rasterećenje zuba, duljina puteva silnica),
   - jaram statora: sa i bez aksijalnih kanala za hlađenje (korak kanala, visine jarma, srednja duljina puta silnice),
   - polovi rotora (prekrivanje pola, širina polnog stopala, razmak između stopala, razmak između polnih jezgri, trapezni pol, površina presjeka polne jezgre, vodljivost rasipnog polja, ekvivalentna duljina puta silnice),
   - idealna duljina: sa i bez radijalnih ventilacijskih kanala,
   - karakteristične veličine (brzina vrtnje na provrtu, broj vodiča i zavoja po fazi i jednoj paralelnoj grani, inducirani naponi, amplituda osnovnog harmonika, nazivni iznos struje, strujni oblog, gustoća struje, ekonomski omjer, itd.);
2) karakteristike praznog hoda:
   - magnetski krug pri nazivnom naponu (faktor osnovnog harmonika indukcije i magnetskog toka u rasporu, magnetski napon raspora, stvarna vrijednost magnetske indukcije u rasporu, magnetski napon zuba statora, magnetski napon jarma statora, magnetski napon statora i zračnog raspora, stupanj zasićenja, faktor β, magnetski tok u zračnom rasporu, rasipni magnetski tok, magnetski tok u polnoj jezgri, magnetski napon pola, ukupni napon magnetskog kruga statora i rotora),
   - magnetski krug za različite stupnjeve napona;
3) proračun armaturnog namota:
   - geometrija (namot s profilnom žicom, namot s okruglom žicom, faktor punjenja utora, težina bakra),
   - električni otpor (radni otpori, reaktancija rasipanja, kapacitivni otpor);
4) proračun uzbude za opterećeno stanje
   - uzbudno protjecanje (uzbudno protjecanje za nazivno opterećenje, protjecanje armature, reakcija armature u poprečnoj osi, magnetski napon statura, reakcija armature u uzdužnoj osi, uzbudno protjecanje za stator i reakcija armature, rasipni magnetski tok pola, ukupni magnetski tok pola, magnetski napon polova, ukupno uzbudno protjecanje),
   - ukupno uzbudno protjecanje za kratki spoj,
   - regulacijske karakteristike;
5) proračun uzbudnog namota:
   - izvedba i dimenzioniranje,
   - višeslojni namot,
   - namot sa okruglom žicom,
   - ostale veličine (radni otpori, uzbudne veličine za prazni hod, uzbudne veličine za kratki spoj, induktivitet, kapacitet, masa);
6) proračun prigušnog namota
   - dimenzije (duljina štapa, površina presjeka štapa i prstena, promjer prstena),
   - protjecanja (amplituda osnovnog harmonika reakcije armature, reakcija armature kod dvofaznog opterećenja, itd.),

- struja (struja u štapu, gustoća struje u štapu),
- radni otpori (lamelirani polovi s prigušnim namotom, masivni polovi),
- masa (štapovi, prsten, ukupno);

7) proračun reaktancija i vremenskih konstanti:
   - reaktancije (koeficijenti oblika polja i magnetskog toka, magnetska vodljivost zračnog raspora, sinkrone reaktancije, prolazna reaktancija, reaktancija rasipanja uzbudnog namota, inverzna reaktancija, nulta reaktancija),
   - vremenske konstante (uzbudnog namota, prigušnog kruga, armaturnog namota);
8) proračun snage i stupnja korisnosti:
   - masa paketa statora (jaram, zubi, ukupno),
   - gubici praznog hoda (jaram statora, zubi statora, krajnje ploče, polno stopalo, ukupno),
   - gubici kratkog spoja (armaturni namot, polno stopalo, zubi statora, kranje ploče, ukupno),
   - gubici u uzbudnom namotu (ukupni gubici, nazivno opterećenje, karakteristike opterećenja),
   - korisnost (gubici ventilacije, količina zraka);
9) proračun masa aktivnih materijala i mehaničke vremenske konstante:
   - masa bakra (armaturni namot, uzbudni namot, prigušni namot),
   - masa željeza (paket statora, paket rotora),
   - zamašna masa,
   - mehanička vremenska konstanta;
10) proračun ostalih detalja kao dopuna osnovnom proračunu:
   - struje kratkih spojeva na stezaljkama armaturnog namota (udarne i trajne),
   - elektromagnetski momenti pri udarnim kratkim spojevima (bazna vrijednost, maksimalne vrijednosti izmjeničnih komponenti, stalne vrijednosti, ukupne vrijednosti),
   - pogon sa dizel motorom (frekvencija osnovnog titrajnog momenta motora, operatorske admintancije, koeficijent sinkronizacijskog momenta, koeficijent prigušnog momenta, sinkronizacijski moment, prigušni moment),
   - jednostrana sila magnetskog privlačenja,
   - demagnetizacija,
   - energija,
   - otpor za demagnetizaciju.

5. Zaključak

U današnjem informatičkom je svijetu problematika dizajna Web orijentiranih višekorisničkih aplikacija vrlo aktualna, a njom se sreće svakodnevno u tisućama primjera: počevši od Web chatova, pa do složenijih Web proračuna. Općenito, takve aplikacije možemo podijeliti u dvije kategorije: a) one koje se izvršavaju isključivo na poslužitelju (*server-side*) i b) one koje se izvršavaju na strani klijenta (*client-side*). Nažalost, nijedan pristup nije potpuno bez mana. Aplikacije koje se izvršavaju isključivo na poslužitelju obično imaju slijedeće karakteristike:

- koje ne idu u prilog dotičnom odabiru:
  - procesorski i resursno su zahtjevne i traže "jake" poslužitelje,
  - imaju problema sa "konkurentnošću" i različitim metodama zaključavanja,
  - predstavljaju težak problem za dizajn arhitekture poslužitelja zbog različitih nivoa opterećenja tijekom dana (vremena vršnog opterećenja rijetka ali opasna za poslužitelj);
- koje idu u prilog dotičnom odabiru:
  - ne zahtijevaju "snažna" računala za klijenta (*thin clients*) - omogućavaju korištenje bilo kakvih verzija Web preglednika (dakle nije nužna Java) i bitno slabijih računala,
  - omogućavaju brz pristup aplikaciji bez potrebe za skidanjem obično velikih (Java) appleta,
  - ne povećavaju značajno mrežni promet do poslužitelja.

S druge strane, aplikacije koje se izvršavaju isključivo na računalu samog korisnika imaju slijedeće karakteristike:

- koje ne idu u prilog dotičnom odabiru:
  - zahtijevaju "snažna" računala (*thick clients*), odnosno svaki korisnik mora imati nešto jače računalo, Web preglednik novije generacije i JVM odnosno instaliranu Javu verzije 1 ili 2,
  - zahtijevaju nešto veće propusnosti mreže i sporije se skidaju preko sporijih mrežnih uređaja (npr. analogni modemi),
  - predstavljaju dodatan problem (sigurnost, itd.) ako applet dohvaća sa poslužitelja dodatne informacije,
  - sporija brzina izvođenja;
- koje idu u prilog dotičnom odabiru:
  - korisnik obično može izvršavati applet i odspojen sa mreže (ne ako applet dohvaća neke dodatne informacije),
  - minimalno opterećenje poslužitelja - poslužitelj se brine samo o ispravnom serviranju appleta, ali ne i proračunu,
  - veća prenosivost i decentralizacija.

Imajući sve ove probleme u vidu, do sada opisana aplikacija je zbog problema sa potrebnom "jačinom" klijenata i mrežnim opterećenjem u potpunosti orijentirana na poslužiteljski način rada sa dodatnom izvedbom i u samostojećem obliku. Nadalje, svi opisani problemi za takav tip aplikacije su uspješno prevaziđeni vrlo pažljivim dizajnom i modernim rješenjima. Aplikacija je (u svim opisanim komponentama) u potpunosti zamišljena kao samostojeće i potpuno rješenje, ne samo što se tiče dizajna s naglaskom na stabilnost, sigurnost i pouzdanost već i prenosivosti kao vrlo važnog faktora. Neke od ideja za poboljšanje se postavljaju kao zanimljive mogućnosti:

- Prebacivanje Perl skripte u izvršni oblik radi bržeg rada - jedna od mogućnosti za izvršavanje ovog je korištenje PerlCC programa koji prevede interpretersku skriptu

u izvršnu datoteku koja koristi Perl bibliotečne pozive.

- Korištenje izračunskog dijela za pokušaj optimiranja stroja genetskim algoritmima: Ideja je napraviti osnovnu liniju strojeva (tzv. gama) s poznatim karakteristikama i stupnjem iskoristivosti i pokušati dobiti stroj traženih svojstava sa što manjom masom (dakle ukupnom cijenom) i što većim koeficijentom korisnosti. Ovo je danas vrlo zanimljiv problem i predstavlja mogući temelj za buduća istraživanje i razvoj.
- Korištenje *daemontoolsa* (http://cr.yp.to/) za nadgledanje ponašanja proračunske aplikacije i njeno praktično stvaranje u vidu *daemona* koji se uvijek automatski pokreće.
- Za dodatnu sigurnost pod Apacheom je moguće koristiti SuExec mogućnost čime nepoželjne ovlasti odbacuje i Perl Web sučelje (umjesto forsiranih vrijednosti u programu), a za dodatnu kontrolu pristupa moguće je koristiti *.htaccess* sa lozinkama te *mod_throttle* za kontrolu brzine pristupa poslužitelju i spomenutom CGI-ju.
- Proširivanje na proračun turbogeneratora [14] i hidrogeneratora velikih snaga.

Naposljetku, ova aplikacija čini doprinos akademskoj zajednici objedinjavanjem nekoliko proračuna za različite izvedbe strojeva koji su do sada bili dokumentirani u dva diplomska rada, [15] i [16] kao i u sveučilišnim priručnicima [13] i [14].

6. Literatura

[1] "ANSI/ISO C++ Professional Programmer's Handbook", 1999.
[2] "Perl Programmers Reference Guide", Perl v5.8.0 dokumentacija, 2003.
[3] Bruce Eckel: "Thinking in C++, 2nd Edition", Vol. 1 & 2
[4] Dr. Dobb's Journal #310, Len DiMaggio: "Testing Unix Daemons", 2000.
[5] GNU "Linux Programmer's Manual"
[6] O'Reilly: "Advanced Perl Programming, 1st edition", 1997.
[7] O'Reilly: "CGI Programming on the World Wide Web, 1st edition", 1996
[8] O'Reilly: "Learning Perl, 2nd edition", 1997.
[9] O'Reilly: "Perl Cookbook, 1st edition", 1998
[10] O'Reilly: "Perl in a Nutshell, 1nd edition", 1998.
[11] O'Reilly: "Practical UNIX & Internet Security, 2nd edition", 1996
[12] O'Reilly: "Programming Perl, 2nd edition", 1996.
[13] R. Wolf: "Osnove električnih strojeva", Zagreb, 1986.
[14] Sirotić-Krajzl: "Upute za proračun sinkronih strojeva", Zagreb, 1972.
[15] Darko Ulaković, "Diplomski rad br. 1427: Elektromagnetski proračun sinhronog hidrogeneratora pomoću računala", Zagreb, 1990.
[16] Đuro Topić, "Diplomski rad br. 1517: Proračun sinkronog generatora snage 9000kVA", Zagreb, 1995.