SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 10??

ZBIRKA POTPROGRAMA ZA PRAĆENJE IZVOĐENJA VIŠEDRETVENOG PROGRAMA

Mentor: prof.dr.sc. LEO BUDIN

LEONARDO JELENKOVIÆ

Mat.br. 36328116
Raèunarska tehnika

ZAGREB, PROSINAC 1996.

SADRŽAJ

1. UVOD

Povijest višedretvenog programiranja poèinje 60-tih, dok se njihova implementacija na UNIX sustavima pojavljuje sredinom 80-tih godina. Prije nekoliko godina stvorena je grupa poznata pod imenom POSIX 1003.4a, koja radi na standardizaciji višedretvenog programiranja. Promatrani operacijski sustav Solaris 2.4 u suštini podržava standard koji predlaže grupa, a koji još nije potpun.

Ideja višedretvenog programiranja jest u tome da se program sastoji od više jedinica koje se samostalno mogu izvoditi. Programer ne mora brinuti o redoslijedu njihova izvoðenja, veæ to obavlja sam operativni sustav. Štoviše, ukoliko je to višeprocesorski sustav, onda se neke jedinice-dretve mogu izvoditi istovremeno.

Atribut višedretveni se ne odnosi na one sustave koji, takoder omoguæuju više dretvi upravljanja nad programom, tako što svaka dretva predstavlja zasebni proces, veæ samo na one sustave koji to omoguæuju unutar jednog procesa, odnosno njegovog adresnog prostora.

Komunikacija među dretvama je jednostavna i brža, jer se obavlja preko zajednièkog adresnog prostora, te se može obaviti bez uplitanja operacijskog sustava.

U ovom radu analiziran je višedretveni sustav Solaris 2.4. Prouèene su moguænosti mjerenja trajanja izvoðenja pojednih dretvi te su napravljene funkcije koje to omoguæuju. Analizirana je upotrebljivost višedretvenosti na sustav zadataka te su navedeni i neki drugi primjeri upotrebe višedretvenosti kod kojih se ona koristi radi ubrzanja.

Svi programi i funkcije su pisani u programskom jeziku C koji se podrazumijeva u ovom okruženju (UNIX). Korišten je operacijski sustav *SunOS 5.5.1* na dvoprocesorskom raèunalu *Ultra-2*.

2. POJMOVI I DEFINICIJE

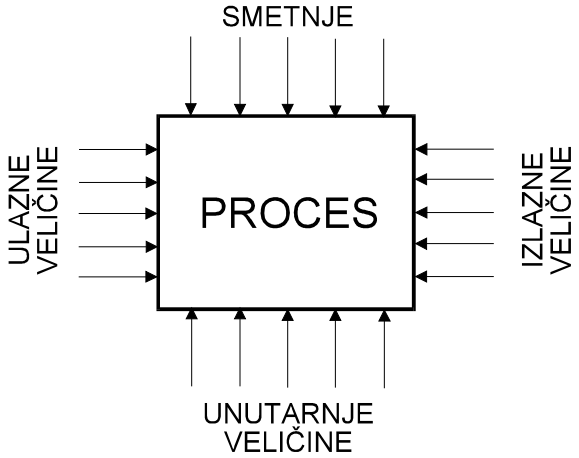
U ovom radu koriste se termini ***proces***, ***dretva*** i ***višedretvenost*** te su isti definirani u ovom poglavlju.

Opæenita definicija pojma ***proces*** glasi:

Proces je vremenski slijed dogaðaja i pojava u sustavu pri kojem se obavlja *pretvorba* i/ili *prijenos* materije i/ili energije i/ili informacije.

Veliène koje opisuju proces mogu se podijeliti na:
- *ulazne veliène* (varijable, parametri, poèetni uvjeti, veliène iz okoline, ...)
- *unutarnje* (varijable stanja)
- *izlazne* (rezultati izvoðenja procesa)
- *smetnje* (utjecaji okoline).

Odnos između tih velièina određuje funkciju procesa. Proces možemo prikazati grafièki kao na slici 2.1.



Slika 2.1 Grafièki prikaz procesa

U konkretnom sluèaju koji se ovdje razmatra, radi se o raèunalnom sustavu u kojem se koriste jedino informacije. Veliène koje opisuju proces su zapravo vrijednosti skupine memorijskih lokacija u raèunalu. Funkciju preslikavanja izmedu ulaznih i izlaznih velièina obavlja skup instrukcija, odnosno, za to određeni program.

Pored ovog apstraktnog znaèenja, pojam ***proces*** se, u ovom radu, veæinom koristi za oznaèavanje osnovnog elementa UNIX (i drugih višekorisnièkih) sustava. ***Proces*** je određen svojim identifikacijskim brojem, adresnim prostorom, prioritetom, korisnikom kome pripada, statusom, signalnom maskom i drugim atributima sustava.

Pojam ***dretva*** ima takoder dva znaèenja, prva upotreba odnosi se na apstraktni pojam, a druga na konkretni element.

Dretva, u apstraktnom znaèenju predstavlja vezu između izvedenih instrukcija programa. Naime, program se obavlja tako da se slijedno izvode njegove instrukcije. Izbor slijedeæe instrukcije ovisi o strukturi programa i o nekoj vrijednosti u programu, međutim, uvijek je samo jedan, tako da se može reæi da instrukcije povezuje jedna ***nit***, odnosno, ***dretva*** upravljanja.

Druga upotreba pojma ***dretva*** odnosi se na element u višedretvenom sustavu, kod kojeg *dretva* predstavlja najmanju nezavisnu cjelinu koja se izvodi nezavisno. Pojam *dretva* je prijevod rijeèi *thread* koja se koristi u istu svrhu u literaturi na engleskom jeziku.

Pod pojmom ***višedretvenost*** podrazumijeva se više dretvi upravljanja nad programom, odnosno, nad izvoðenjem njegovih instrukcija. Instrukcije istog programa upotrebljavaju se u više nezavisnih izvoðenja, odnosno, svako ima vlastiti tok izvoðenja. Engleski termin za ovaj pojam jest *multithreading*.

3. VIŠEDRETVENI OPERACIJSKI SUSTAV SOLARIS 2.4

3.1 Osnovno o višedretvenosti

Kao što i sam naziv govori, višedretveni sustav jest sustav koji podržava više dretvi upravljanja nad programom. Bez dodatnog komentara moglo bi se doæi do krivog zaključèka da su svi operacijski sustavi kojima je osnovna jedinica proces, također višedretveni, jer jedan program može u izvođenju stvoriti više procesa, odnosno, dretvi upravljanja. Međutim, višedretvenost se ne odnosi na takve sustave, veæ samo na one sustave koji omoguæuju više dretvi upravljanja nad programom unutar jednog procesa, odnosno, unutar njegova adresna prostora.

Prednost takvih sustava prvenstveno je u tome što zauzimaju mnogo manje sredstava sustava. Razlog tome jest što stvaranje novog procesa zahtijeva i stvaranje (rezerviranje) novog adresnog prostora, dok stvaranje nove dretve to ne zahtijeva. Navedeni razlog se također odražava na vrijeme stvaranja, koje je kod stvaranja dretvi èak i do tisuæu puta manje nego kod stvaranja novog procesa.

Zbog toga što dijele adresni prostor istog procesa, međusobna komunikacija i sinkronizacija među dretvama mnogo je brža od one među procesima. Promjena nekog podatka od strane jedne dretve istovremeno postaje vidljiva svima ostalima, omoguæujuæi tako razmjenu podataka bez uplitanja operacijskog sustava.

Istovremenost pristupa podacima od strane više dretvi se ponekad ne smije dopustiti. To su situacije u kojima dretve koriste zajedničke varijable, a èija bi istovremena upotreba (i eventualno promjena) izazvala greške. Takvi se kritièni odsjeèci programa zaštiæuju zaključavanjem. Kod korištenja zajedničkih podataka treba također uzeti u obzir da promjena koju uzrokuje jedna dretva ne mora baš istog trenutka biti svima vidljiva, odnosno, zbog toga što procesori koriste vlastite priručne memorije, promjena podatka se privremeno može obaviti samo lokalno, a ažuriranje glavne memorije može nastati tek kasnije. Takvi se dijelovi programa također zaštiæuju zaključavanjima i iz razloga što te funkcije zaključavanja/otključavanja također ažuriraju glavnu memoriju.

Promatrani sustav razlikuje dvije kategorije dretvi. Prva kategorija predstavlja *korisničke* dretve, koje su i razmatrane u ovom radu. Druge, *sistemske*, spadaju u područje sistemskih programera te one nisu razmatrane u ovom radu.

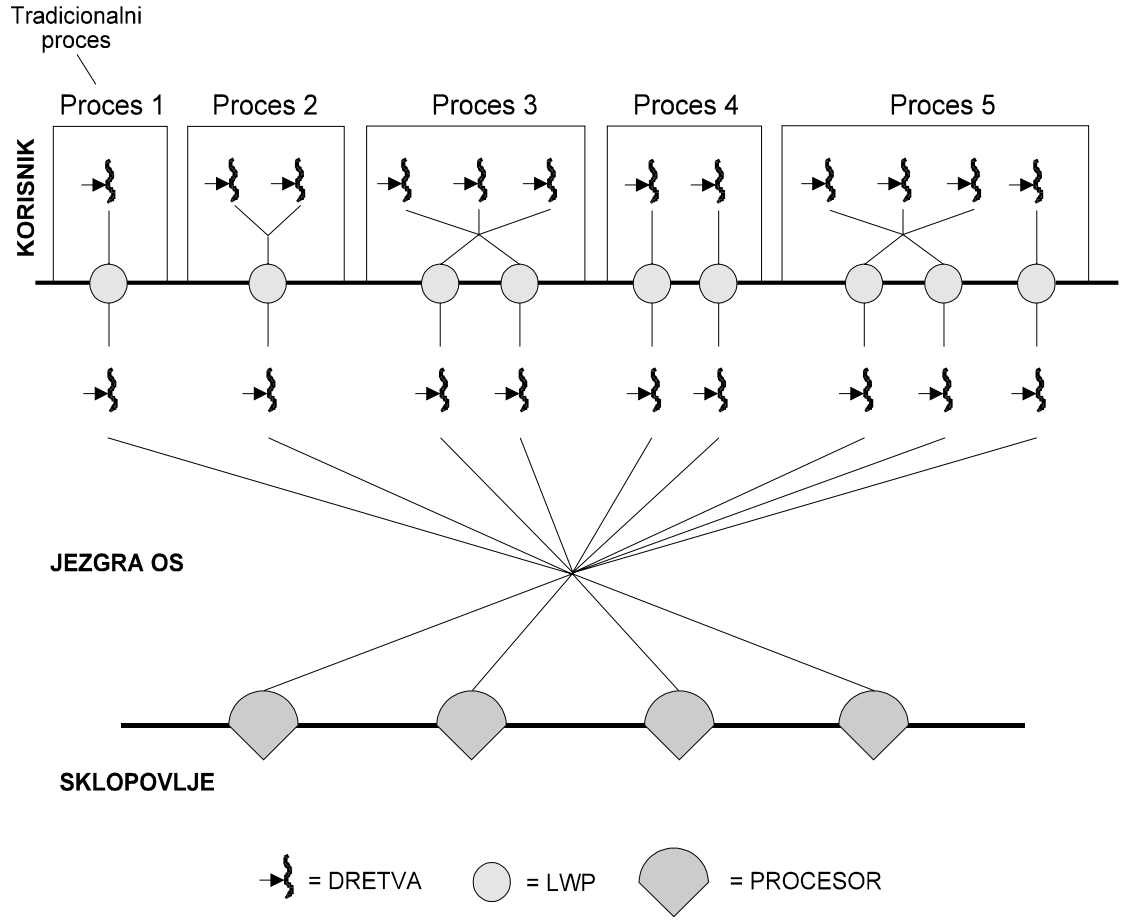
3.2 Arhitektura višedretvenog sustava

Vidljivost dretvi jest samo unutar procesa, èiji su one dio i èija sredstva dijele (adresni prostor, otvorene datoteke, ...).

Slijedeæa stanja su, međutim, jedinstvena svakoj dretvi:

- identifikacijski broj dretve
- registarsko stanje, uključujuæi programsko brojilo i kazaljka stoga
- stog
- signalna maska
- prioritet
- privatni prostor same dretve.

Upravljanje dretvi, odnosno, osiguravanje da se sve dretve izvode, obavlja dretvena biblioteka ***libthread***. Upravljanje se obavlja na korisnièkoj razini, a ne na razini operativnog sustava. Veza izmedu dretve i jezgre su tzv. ***lightweight*** procesi, za koje se u daljnjem tekstu koristi oznaka LWP. LWP jest osnovna dretva upravljanja na razini jezgre sustava. LWP se, sa strane programera, može predoèiti jednostavno kao virtualni procesor. Arhitektura višedretvenog sustava *Solaris 2.4* prikazana je na slici 3.1.



Slika 3.1 Arhitektura višedretvenog sustava

Iz slike je vidljivo da standardni UNIX procesi imaju samo jedan LWP, odnosno, jednu dretvu upravljanja. Također, svaka dretva ne mora imati vlastiti LWP, veæ ih više njih može koristiti iste LWP-ove, a o èemu brine *libthread*.

Razlikujemo dvije skupine dretvi:

- ***vezane*** (engl. *bound threads*)
- ***nevezane*** (engl. *unbound threads*).

Prve, ***vezane***, su takve dretve kojima je pridjeljen vlastiti LWP koji izvodi samo njene instrukcije. Upravljanje tih dretvi indirektno obavlja sam sustav upravljajuæi dotièni LWP. To jest uzrokom nešto sporije komunikacije među takvim dretvama, jer se ona obavlja uz uèešæe sustava.

Druge, ***nevezane***, nisu vezane za vlastiti LWP, veæ se mogu izvoditi na bilo kojem LWP-u koji se nalazi na raspolaganju procesu, a nije vezan za jednu dretvu. Pri izvoðenju takve dretve, LWP koji ju izvodi poprima stanje te dretve (instrukcije, signalnu masku, stog, ...).

U veæini sluèajeva dovoljno je koristiti nevezane dretve koje manje optereæuju sustav.

Razlogom korištenja vezanih dretvi treba biti jedan od slijedeæih:

- globalno rasproreðivanje dretvi
- dodjela alternativnog signalnog stoga dretvi
- korištenje vremenskih brojaèa pripadajuæih LWP-a.

Prvi razlog jest zapravo taj, da tada sustav rasporeðuje dretve, odnosno, pripadajuæe LWP-ove na raspoložive procesore, izbjegavajuæi tako promjene konteksta nad samim LWP-om koje nastaje u sluèaju nevezanih dretvi, što se odražava na brzinu izvoðenja. Takvi su sluèajevi kada ima više dretvi nego LWP-ova. Vezane dretve se, meðutim, sporije sinkroniziraju. Kombinacija vezanih i nevezanih dretvi jest ponekad najbolje riješenje, koristeæi tako prednosti jednih i drugi dretvi.

Dretva, koja inaèe nije vezana za LWP, prilikom izvoðenja sistemskog poziva, vezuje se na dotièni LWP na kome se izvodi i ostaje vezana sve dok se sistemska rutina ne obavi.

Svaka dretva ima vlastitu signalnu masku, meðutim, ponašanje dretvi prilikom razlièitih signala ovisi o vrsti signala. Uzrok signala može biti izvan procesa ili unutar njega, može se odnositi samo na određenu dretvu, određeni LWP ili pak na cijeli proces. Više o ovom problemu može se pronaæi u literaturi [1] te u uputama samog sustava (*man* stranicama).

3.3 Osnovne funkcije za rad sa dretvama

U ovom poglavlju su prikazane samo najosnovnije funkcije, odnosno, sistemski pozivi za rad sa dretavama. Ostale se mogu naæi u literaturi [1] i *man* stranicama sustava. Veæina standardnih funkcija poprima ponešto drukèije ponašanje u višedretvenim programima (korištenje zastavice _REENTRANT) radi njihovog ispravnog rada. Naime, radi paralelnog rada više dretvi, neke funkcije koje koriste globalne podatke procesa mogu raditi neispravno. To se posebno odnosi na rad sa datotekama, generatorima sluèajnih brojeva, ispisom na standardne izlaze (zaslon terminala).

Prema ponašanju u višedretvenim programima, funkcije možemo podijeliti na:

- *pouzdane* - rade ispravno u višedretvenim programima
- *pouzdane sa iznimkama* -u iznimnim sluèajevima može doæi do grešaka
- *nepouzdane* - u višedretvenim programima može doæi do grešaka
- *MT-pouzdane* - posebno pripremljene za višedretvene programe
- *MT-pouzdane sa iznimkama*
- *asinkrono pouzdane* - nema potpunog zastoja dretve sa samom sobom kada su one prekinute u izvoðenju ovih funkcija.

Obièno se u opisu funkcije (*man* stranice) navodi kategorija funkcija te dodatne napomene o primjeni u višedretvenim programima. Funkcije prireðene posebno za višedretvene programe imaju obièno nastavak *_r* , ili zapoèinju sa *thr*.

3.3.1 Stvaranje dretvi

Sve dretve, osim prve, inicijalne, koja nastaje stvaranjem procesa, nastaju pozivom ***thr_create***.

Opæi oblik funkcije jest:

int ***thr_create****(void **sp**, size_t **ss**, void * (***poèetna_f**) (void *), void ***arg**,*
long ***zast**, thread_t ***nova_dr**);*

Parametri:

- ***sp*** - sadrži poèetnu adresu stoga, ako nije NULL
- ***ss*** - velièina stoga (u bajtovima), ako nije nula
- ***pocetna_f*** - kazaljka na poèetnu, prvu funkciju èije instrukcije stvorena dretva poèinje izvoditi
- ***arg*** - kazaljka na argument koji se prenosi u stvorenu dretvu, ukoliko nije NULL
- ***zast*** - skup bitova koji određuju karakteristike stvorene dretve
- ***nova_dr*** - kazaljka na lokaciju gdje se sprema identifikacijski broj stvorene dretve

Velièine *sp* i *ss* najbolje je postaviti na NULL i 0, odnosno omoguæiti dretvenoj biblioteci *libthread* da rezervira stog stvorenoj dretvi.

Stvorenoj dretvi može se prenijeti samo jedan argument, odnosno jedna kazaljka. Ukoliko je potrebno prenijeti više parametara, onda se oni mogu grupirati u strukturu te se šalje njena kazaljka.

Vrijednost *zast* obièno se dobiva kao rezultat operacija logièkog zbrajanja između više razlièitih vrijednosti, od kojih svaka predstavlja neki atribut. Te vrijednosti mogu biti kombinacija slijedeæih konstanti:

- THR_BOUND - stvara vezanu dretvu
- THR_DAEMON - oznaèujemo dretvu kao *daemon*, nevidljivu ostatku procesa
- THR_DETACHED - dretve stvorene ovom zastavicom možemo “reciklirati” pošto ove završe sa radom, odnosno, možemo iskoristi njihova sredstva za nove dretve
- THR_NEW_LWP - ovom zastavicom poveæavamo broj LWP-a za jedan
- THR_SUSPENDED - dretva stvorena ovom zastavicom poèinje izvoðenje tek pošto je pokrenuta pozivom *thr_continue* od strane neke druge dretve.

Najjednostavniji poziv funkcije jest:

thr_create*(NULL, 0, **poèetna_f**, NULL, 0, NULL);*

koji stavara nevezanu dretvu i koji je u veæini sluèajeva najbolje riješenje. Nakon ispravnog završetka funkcija vraæa nulu.

3.3.2 Završetak rada dretve

Normalan završetak dretve jest njen izlazak iz prve, inicijalne funkcije, ili pozivom funkcije ***thr_exit***.

Opæi oblik funkcije:

*int **thr_exit**(void ***status**);*

Parametar:

- ***status*** - kazaljka na stanje sa kojim dretva završava.

Ukoliko dretva nije stvorena uz zastavicu THR_DETACHED, tada se broj dretve i njen izlazni status zadržava sve dok ona nije doèekana ili do kraja procesa. Inaèe se izlazno stanje ignorira te se dretva može "reciklirati", odnosno, mogu se iskoristiti njena sredstva za stvaranje nove dretve. Nakon ispravnog završetka funkcija vraæa nulu.

Dretva èeka na završetak druge dretve pozivom funkcije ***thr_join***.

Opæi oblik funkcije:

*int **thr_join**(thread_t **èekana_dr**, thread_t **doèekana_dr**, void ****stanje**);*

Parametri:

- ***èekana_dr*** - identifikacijski broj dretve na èiji se kraj èeka, ukoliko nije nula, kada se èeka nazavršetak bilo koje dretve
- ***doèekana_dr*** - kazaljka na broj dretve koja je završila i doèekana je ovom dretvom, ukoliko nije NULL
- ***stanje*** - kazaljka na kazaljku izlaznog statusa doèekane dretve.

Funkcija ***thr_join*** zaustavlja izvoðenje pozivajuæe dretve sve dok određena dretva ne završi sa radom. Nakon ispravnog završetka funkcija vraæa nulu. Slijedeæim dijelom programa može se èekati na sve dretve koje nisu stvorene zastavicama THR_DAEMON i THR_DETACHED:

*while(**thr_join**(0, NULL, NULL)==0) ;*

Normalni završetak višedretvenog programa zbiva se kada sve dretve završe sa radom, odnosno, kada prva, poèetna dretva izaðe iz prve funkcije (*main*). Prijevremeni završetak zbiva se pozivom funkcije ***exit*** od strane bilo koje dretve, ili pak nekim vanjskim signalom (SIGKILL, SIGSEGV, SIGINT, SIGTERM, ...). Kada sve dretve koje nisu stvorene zastavicom THR_DAEMON završe, automatski završavaju i takve dretve.

3.3.3 Određivanje broja LWP-a

Broj LWP-a može se poveæavati zastavicom THR_NEW_LWP prilikom stvaranja dretvi, ali također i pozivom funkcije ***thr_setconcurrency***.

Opæi oblik funkcije:

*int **thr_setconcurrency**(int **nova_razina**);*

Parametar:

- ***nova_razina*** - željeni broj LWP-a u procesu, ne raèunajuæi LWP-ove vezanih dretvi.

O rasporedu nevezanih dretvi na postojeæe LWP-ove brine dretvena biblioteka *libthread*. Programer tu može utjecati tako da dretvama pridruži razlièite prioritete (*thr_getprio, thr_setprio*) te pozivom određenih funkcija (*thr_yield*).

3.3.4 Međusobna komunikacija i sinkronizacija

Promatrani operacijski sustav omoguæuje èetiri naèina sinkronizacije među dretvama:

- *međusobno isključèivanje*
- *uvjetne varijable*
- *zaključavanja èitaj/piši*
- *semafori.*

U nastavku su prikazana samo prva dva naèina sinkronizacije, odnosno, oni koji su korišteni u programima vezanim uz ovaj rad.

Kada je potrebno zaštititi neke dijelove programa od istovremenog korištenja više dretvi (kritièni odsjeèci), tada se koriste funkcije **međusobnog isključèivanja**, odnosno, dijelovi programa se zaključuju pomoæu kontrolnih varijabli - ključèeva. Zaključavanje se obavlja funkcijom ***mutex_lock***, a otključavanje funkcijom ***mutex_unlock***.

Opæi oblici funkcija:

int ***mutex_lock****(mutex_t **ključ);*
int ***mutex_unlock****(mutex_t **ključ);*

Parametar:

- ***ključ*** - kazaljka na varijablu zaključavanja.

Sve dretve, koje pokušaju zaključati veæ zaključanu varijablu ostaju blokirane na pozivu sve dok varijabla ostaje zaključana. Kada se varjabla otključa, onda samo jedna dretva ulazi slijedeæa u kritièni osjeèak, uz zaključavanje varijable.

Zaključavanjem se smanjuje moguæa paralelnost u izvođenju skupine dretvi, te je njihova upotreba prihvatljiva (obavezna) samo u kritičnim odsjeècima.

Prije korištenja, varijable međusobnog isključivanja potrebno je inicijalizirati pozivom ***mutex_init***.

Uvjetne varijable se koriste kada želimo da neka dretva zaustavi svoje izvođenje te èeka da se određeni uvjet ispuni, tj. da ga ispuni neka druga dretva. Funkcijama za korištenje uvjetnih varijabli obavezno prethodi zaključavanje, pošto su uvjetne varijable globalne, zajedničke cijelom procesu.

Osnovne funkcije za rukovanje sa uvjetnim varijablama su ***cond_wait***, ***cond_signal*** i ***cond_broadcast***.

Opæi oblici funkcija:

int ***cond_wait****(cond_t **uvjet, mutex_t **ključ);*

int ***cond_signal****(cond_t* ****uvjet****);*
int ***cond_broadcast****(cond_t* ****uvjet****);*

Parametri:

- ***uvjet*** - kazaljka na uvjetnu varijablu
- ***kljuè*** - kazaljka na varijablu zakljuèavanja.

Pozivom ***cond_wait*** pozivajuæa dretva se postavlja u stanje èekanja koje završava neka druga dretva pozivima ***cond_signal*** i ***cond_broadcast***. Poziv *cond_signal* ispunjuje uvjet za nastavaka samo jedne dretve iz reda èekanja na istu uvjetnu varijablu, dok poziv *cond_broadcast* omoguæuje svim takvim dretvama nastavak izvoðenja. Ako niti jedna dretva nije bila blokirana na uvjetnoj varijabli prilikom poziva *cond_signal* i *cond_broadcast*, onda ovi pozivi nemaju nikakav uèinak, tj. ako u slijedeæem trenutku neka dretva pozove *cond_wait* ona ostaje blokirana.

Prije korištenja, uvjetne varijable je potrebno inicijalizirati pozivom ***cond_init***.

4. MJERENJE VREMENA IZVOĐENJA VIŠEDRETVENOG PROGRAMA

4.1 Naèini mjerenja vremena izvoðenja

Operacijski sustav sam vodi vremensku statistiku o izvoðenju procesa, te ako je takvo ukupno potrošeno vrijeme sustava na određenom dijelu nekog programa dovoljno, onda se to vrijeme može dobiti pomoæu nekih sistemskih poziva. Najjednostavniji jest pozivom funkcije ***clock*** koja vraæa broj mikrosekundi potrošenih na sredstvima sustava od prvog poziva te funkcije. *clock* daje sumu potrošenog korisniènog vremena i vremena sustava potrošenog na raèun procesa, a ne spada u samo izvoðenje. U veæini sluèajeva jednodretvenih, klasiènih programa, ovaj je poziv dostatan.

To, međutim, nije tako sa višedretvenim programima. Funkcija *clock* upotrijebljena u višedretvenom programu, također daje sumu korisniènog i sistemskog vremena, ali je to vrijeme zbroj vremena potrošenog na sve dretve. U sluèaju višeprocesorskih sustava to je vrijeme obièno èak i veæe od stvarno proteklog vremena.

Kod takvih programa želimo izmjeriti trajanja pojedinih dretvi. Dretva nema moguænosti mjerenja vremena, ali zato može koristiti brojila pripadajuæih LWP-ova. Svaki LWP ima vlastite brojaèe od kojih svaki odbrojava određeno vrijeme. Tako postoje dva brojila stvarnog vremena, brojilo korisniènog potrošenog vremena te brojilo ukupnog potrošenog vremena sustava. Brojila odbrojavaju od neke poèetne vrijednosti do nule, kada šalju odgovarajuæe signale pripadajuæim LWP-ovima. Uvjet, dakle, koji dretva mora ispunjavati da bi koristila navedena brojila, jest da mora biti vezana za vlastiti LWP. Ukoliko se koriste brojila LWP-a, a dretve nisu vezane za njega tada su dobivena vremena suma vremena svih dretvi koje su se u međuvremenu izvršavale koristeæi taj LWP.

Pored tih brojila, svaki LWP se može vremenski analizirati pozivom ***profil***. Na svaki otkucaj sistemskog sata, jedno brojilo u skupu od više njih, biva poveæano za jedan. Koje, ovisi o trenutnoj vrijednosti programskog brojila. Tako se može dobiti dijagram potrošnje vremena pojedine dretve s obzirom na programski kod koji ona izvodi. Nedostatak takvog brojila leži u tome što se brojaèi poveæavaju samo na otkucaj sata, te ukoliko se dotièna dretva ne nalazi u tom trenutku u izvoðenju, onda se gubi taj otkucaj, tj. profil daje kvalitativnu, ali ne i kvantitativnu sliku. To posebno dolazi do izražaja u sustavima sa velikom periodom sistemskog sata,kao u promatranom sustavu.

4.2 Potprogrami za mjerenje trajanja pojedinih dretvi

Trajanje se može mjeriti samo u sluèaju *vezanih* dretvi, odnosno, tada se može dobiti vremena vezana za rad određene dretve, te se sve navedeno odnosi na takve dretve.

Navedeni potprogrami za mjerenje vremena koriste dva poziva za rad sa brojilima LWP-ova, a to su ***getitimer*** i ***setitimer*** (engl. *get/set interval timer*).

Opæi oblici funkcija:

*int **getitimer**(int **koji**, struct itimerval ***vrijednost**);*
*int **setitimer**(int **koji**, const struct itimerval ***vrijednost**, struct itimerval ***stara_vr**);*

Parametri:

- ***koji*** - određuje brojilo nad kojim se obavlja operacija
- ***vrijednost*** - kazaljka na strukturu koja sadrži podatke o velièini intervala te o preostalom vremenu (sekunde i mikrosekunde)
- ***stara_vr*** - kazaljka na strukturu u koju se smještaju podaci o brojilu (interval, preostalo vrijeme) prilikom postavljanja drugih.

Svaki LWP raspolaže sa èetiri brojila:

- ITIMER_REAL - odbrojava u stvarnom vremenu
- ITIMER_VIRTUAL - odbrojava samo prilikom izvoðenja pripadnog LWP-a
- ITIMER_PROF - odbrojava prilikom izvoðenja, te također kada sustav obavlja neke akcije u korist tog LWP-a
- ITIMER_REALPROF - odbrojava u stvarnom vremenu i posebno je namijenjen analiziranju višedretvenih programa, slièno pozivu *profil.*

4.2.1 Inicijalizacija vremenskih brojila

Za mjerenje trajanja dretvi koriste se samo tri brojila: ITIMER_REAL, ITIMER_VIRTUAL i ITIMER_PROF. Pokretanje svih navedenih brojila obavlja funkcija ***thr_inittimers***, odnosno, ***realtime_init***, ***usertime_init*** i ***clock_init*** pokreæu pojedina brojila.

Opæi oblici funkcija:

*int **thr_inittimers**();*
*int **realtime_init**();*
*int **usertime_init**();*
*int **systime_init**();*

Funkcije postavljaju poèetne vrijednosti brojila. Sa stanovišta korisnika ove funkcije postavljaju brojila na nulu, dok je u stvarnosti obrnuto. Brojila se postavljaju na neku veliku vrijednost od koje odbrojavaju. Korisnik pomoæu narednih funkcija dobiva proteklo vrijeme te on za ovu internu reciproènost ne mora niti znati.

Nakon uspješne inicijalizacije funkcije vraæaju na nulu, dok nakon neke greške vraæaju -1.

4.2.2 Dobivanje proteklog vremena

Funkcije kojima se dobiva proteklo vrijeme jesu: ***thr_realtime***, ***thr_usertime*** i ***thr_clock***.

Opæi oblici funkcija:

*long **thr_realtime**(struct timeval ***vrijeme**);*
*long **thr_usertime**(struct timeval ***vrijeme**);*
*long **thr_clock**(struct timeval ***vrijeme**);*

Parametar:

- ***vrijeme*** - kazaljka na strukturu gdje se pohranjuje proteklo vrijeme u sekundama i mikrosekundama, ukoliko je razlièito od NULL.

Funkcija ***thr_realtime*** vraæa stvarno vrijeme proteklo od pokretanja brojila (*thr_inittimers, realtime_init*) do samog poziva funkcije.

Funkcija ***thr_usertime*** vraæa vrijeme koje je dretva provela u izvoðenju na procesorima sustava, ne ukljuèujuæi sistemske pozive.

Funkcije ***thr_clock*** vraæa vrijeme koje je sustav potrošio na izvoðenje i na ostale akcije u vezi pozivajuæe dretve.

U sluèaju greške funkcije vraæaju -1. Prije poziva ovih funkcija, brojila moraju biti inicijalizirana pozivom ***thr_inittimers***, odnosno, pozivima ***realtime_init***, ***usertime_init*** i ***systime_init***. Ako to nije uèinjeno, dobivene vrijednosti ovih funkcija nemaju realnog znaèenja.

Ispisi funkcija nalaze se u prilogu.

4.2.3 Upotreba potprograma

Potprogrami su kompilirani te je od njih napravljena biblioteka pod nazivom ***thrtimers*** (*libthrtimers.a*). Zaglavlja su definirana u datoteci *thrtimers.h* tako da ju je pri korištenju navedenih potprograma potrebno uključiti u program.

Povezivanje biblioteke *thrtimers* sa ostalim kompiliranim dijelovima programa postiže se na slijedeæi naèin:

cc[1] -x y[2] -L/*put_do_datoteke*/ -l*thrtimers* ,

odnosno, sa zastavicom **L** navodimo put, a sa **l** ime biblioteke bez poèetna tri slova (*lib*) i nastavka (*.a*).

[1]C-kompilator, može i bilo koji drugi kompilator ili povezivaè (*linker*)
[2]x i y predstavljaju sve ostale potrebne zastavice i parametre

5. UTJECAJ VIŠEDRETVENOG PROGRAMA NA OPTEREÆENJE SUSTAVA

Kao što je veæ i ranije reèeno, višedretveni program u izvoðenju koristi adresni prostor jednog procesa te je tako mnogo prihvatljiviji od višeprocesnog programa, sa stanovišta zauzeæa sredstava sustava.

Sa stanovišta procesorskog optereæenja, utjecaj višedretvenog programa ovisi o broju LWP-ova koje program koristi. LWP je osnovna dretva na razini operacijskog sustava, odnosno, jedinica koju sustav raspodijeljuje na svoje procesore. Rasporeðivanje se obavlja po nekom algoritmu, koji u osnovi predstavlja podjelu vremena. Odnosno, sustav dodjeljuje svaki LWP nekom procesoru na određeno vrijeme, te kada to vrijeme istekne, ili se desi neki sistemski poziv, procesoru se oduzme trenutni LWP te mu se dodijeli slijedeæi ili obavlja neke radnje vezane uz sustav.

Sustav ne optereæuju dretve kao takve, veæ LWP-ovi na kojima se one izvode. Što ih program više koristi, to je i veæi njegov udio u svim LWP-ovima sustava, odnosno, više optereæuje sustav.

Mjera za optereæenost sustava jest prosjeèan broj poslova koji èekaju u redu za izvoðenje. Na UNIX operativnom sustavu optereæenje se može dobiti naredbama *uptime* i *w* koje daju podatke o optereæenosti u zadnjoj minuti, zadnjih pet minuta i zadnjih petnaest minuta. Broj poslova predstavlja broj procesa u tradicionalnom UNIX-u, odnosno, broj LWP-ova u razmatranom sustavu, koji èekaju na izvršenje. Ako je broj manji od jedan (od 0 do 1), tada je sustav slabo optereæen (empirijski podaci). Ukoliko LWP izvodi program koji obavlja neka proraèunavanja, tada taj LWP konstantno optereæuje sustav za vrijeme svog postojanja. Ukoliko je to neki uslužni program koji obavlja određene kratke akcije na zahtjev korisnika (editor, komandna ljuska, ...) tada taj LWP optereæuje sustav samo u kratkim trenucima (kada korisnik pritisne bilo koju tipku) te vrlo malo optereæuje sustav.

Pošto se u ovom radu promatraju programi koje bi višedretvenost trebala ubrzati, odnosno, radi se o vremenski zahtjevnijim matematièkim proraèunavanjima, upotreba takvih programa prilièno optereæuju sustav i to proporcionalno broju LWP-ova na kojima se izvode.

Ukoliko je sustav vrlo malo optereæen, tada se preporuèuje korištenje manjeg broja LWP-ova (onoliko koliko ima procesora) jer se time izbjegava nepotrebno vrijeme promjena konteksta nad procesorima (promjena LWP-a).

Ukoliko je sustav više optereæen, trajanje ovisi o tome koliko vremena sustav dodijeljuje programu, tj. njegovim LWP-ovima. Program æe se tada brže obaviti uz veæi broj LWP-ova, pod uvjetom da se program dade tako rastaviti. Optereæenje sustava, naravno, još æe više time porasti te treba također o tome voditi raèuna. Naime, ako se time znatno poveæa optereæenost sustava, sustav postaje prespor te je rad na njemu nemoguæ. Npr. ako program stvori 1000 vezanih dretvi tada æe i ostali korisnici osjetiti optereæenost sustava, tako što æe prièekati i preko minute prije nego æe im se na zaslonu pojaviti znak koji su pritisnuli na tastaturi.

Ako se program sastoji samo od vezanih dretvi tada je optereæenje koje uzrokuje program proporcionalno broju dretvi. Inaèe, ako program koristi i nevezane dretve, tada

optereæenje ne ovisi o ukupnom broju dretvi veæ o ukupnom broju LWP-ova na kojima se one izvode.

Višedretveni program, dakle, može i bitno opteretiti sustav veæim brojem LWP-a na kojima se izvode dretve veæih proraèuna ili drugih operacija koje ne zahtijevaju èesto uèešæe korisnika.

6. TRAJANJA NEKIH TIPIÈNIH OPERACIJA SA DRETVAMA

U ovom poglavlju su navedena prosjeèna vremena koja su potrebna za stvaranje novih dretvi uz razlièite atribute, a također i vremena međusobne sinkronizacije. Radi ilustracije navedena su i vremena izvođenja tipiènih matemarièkih operacija.

Mjerenja su uglavnom sprovedena pri malim optereæenjima sustava tako da bi pri prosjeènom optereæenju, odnosno pri veæim optereæenjima ta vremena bila nešto veæa, pogotovo realno vrijeme, dok bi korisnièko i sistemsko ostalo približno isto.

6.1 Trajanje stvaranja dretvi

Stvaranje nove dretve zahtjeva stvaranje (rezervaciju) stanja pridruženih dretvi: identifikacijski broj, registarsko stanje, stog, signalna maska te prioritet. Ukoliko stavranju dretve ne slijedi i stvaranje novog LWP-a, odnosno, zastavice THR_NEW_LWP i THR_BOUND nisu postavljene, tada se nova dretva stvara bez utjecaja jezgre sustava, a uz uèešæe biblioteke *libthread*. Stvaranje LWP-ova obavlja jezgra sustava te ova operacija zahtjeva nešto više vremena.

U tablici 6.1.1 su prikazani rezultati mjerenja stvaranja dretvi.

Tip dretve	Stvaranje (μs)	Dodjela konteksta(μs)	Èekanje na kraj dretve(μs)
Nevezane	320	40	120
Vezane	400	50	100
Nevezane *reciklirane*	150	40	-[3]
Vezane *reciklirane*	230	50	-

Tablica 6.1.1 Vremena stvaranja dretvi

Navedeni podaci u tablici ne smiju se uzeti kao apsolutni. Naime, zbod toga što je trajanje navedenih operacija puno manje od razluèivost sistemskog sata, bilo je potrebno stvoriti veæi broj dretvi (500, 1000). Pri takvom broju dretvi optereæenje nije normalno. Kod nevezanih dretvi optereæenje pada u korisnièko područje (biblioteke *libthread*), a kod vezanih, na sustav. Dobivena vremena pokazuju znatno raspršenje oko srednjih vrijednosti, navadenih u tablici. Vremena stvaranja novih LWP-ova, ovakvim mjerenjem ispadaju višestruko veæa od stvaranja vezanih dretvi, pri èemu se također stavaraju LWP-ovi, te ta vremena nisu ni unesena u tablicu, jer nemaju smisla.

Iz tablice bi se moglo, usporedbom vremena stvaranja vezanih i nevezanih dretvi, odrediti neko okvirno vrijeme stvaranja LWP-a od 80 μs. Za vrijeme inicijalne dodjele konteksta, mjerenje daje 40, tj. 50 μs. Vrijeme promjene konteksta se također kreæe u tim granicama.

[3] ne može se èekati na kraj dretvi koje su stvorene zastavicom THR_DETACHED

Pri normalnoj upotrebi višedretvenosti ta se vremena mogu zanemariti. Ukoliko se pak, dretve upotrijebljavaju tako da se uèestalo stvaraju za određene zadatke, tada treba raèunati sa tim vremenom stvaranja. Ukoliko su zadaci jednostavni, tj. kratko traju, vremena stvaranja æe biti dominantna, što æe rezultirati sporijem obavljanju od slijednog. U tom sluèju bolje je imati stalno aktivno više dretvi koje obavljaju potrebne zadatke, upravljane jednom, glavnom dretvom. Ukoliko su zadaci složeniji, tj. traju barem za red velièine više od vremena stvaranja dretvi, tada se mogu stvariti dretve kada su potrebne. U ovakvom sluèaju poželjno je dretve stvarati sa zastavicom THR_DETACHED, tako da se pri stavaranju novih, koriste sredstva prethodnih, koje su završile s radom, smanjujuæi vrijeme stavranja i do 50%. Pri stvaranju dretvi sa ovom zastavicom, prvo stvaranje traje normalno dugo, dok ostala traju kraæe ako je neka dretve stvorena ovom zastavicom završila.

6.2 Trajanje sinkronizacije uvjetnim varijablama

Najbrža i najjednostavnija sinkronizacija među dretvama postiže se upotrebom uvjetnih varijabli. Mjerenje trajanja sinkronizacije uèunjeno je upotrebom dvije dretve. Promatrana je sinkronizacija među vezanim dretvama, nevezanim dretvema sa samo jednim LWP-om i sa dva LWP-a. Trajanja su prikazana u tablici 6.2.1.

Tip dretvi	Broj LWP-a	1000 iteracija		10000 iteracija		prosjek	
		t_{REAL} (s)	$t_{USER+SYS}$ (s)	t_{REAL} (s)	$t_{USER+SYS}$ (s)	t_{REAL} (μs)	$t_{USER+SYS}$ (μs)
Nevezane	1	0.024	0.025	0.243	0.24	24.2	24.5
Nevezane	2	0.094	0.11	1.42	1.35	118	123
Vezane	(2)	0.088	0.13	0.839	1.09	86	120

Tablica 6.2.1 Vremena sinkronizacije

Sinkronizacija je najbrža kod nevezanih dretvi sa samo jednim LWP-om dok je sinkronizacija nevezanih dretvi na dva LWP-a (koliko inaèe ima i procesora) najsporija. Međutim, nevezane dretve sa samo jednim LWP-om se ne mogu izvoditi paralelno te se njima ne može postiæi nikakvo ubrzanje na višeprocesorskom sustavu. Nasuprot njima, vezane dretve, koje se oko 3.5 puta sporije sinkroniziraju, mogu vrlo efikasno iskoristiti višeprocesorski sustav, te su one pravi izbor, ukoliko se radi na ubrzanju programa. Upotreba nevezanih dretvi prvenstveno je vezana za one aplikacije kojima brzina obavljanja nije toliko bitna, veæ se upotrebom višedretvenosti postiže odvajanje nezavisnih dijelova programa, koji rade sasvim druge zadatke (npr. svaka dretva predstavlja neki prozor unutar istog procesa, koja se aktivira na zahtjev korisnika).

6.3 Usporedba vremena stvaranja, sinkronizacije i matematièkih operacija

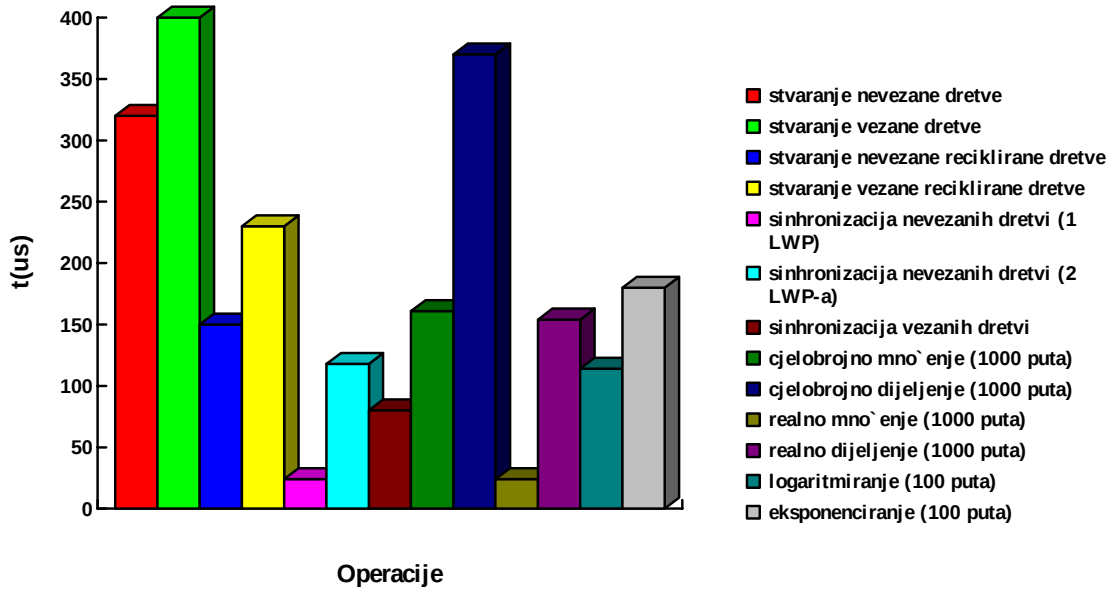
Prosjeèna vremena obavljanja matematièkih operacija nalaze se u tablici 6.3.1.

Operacija	(int) ×	(int) :	(double) ×	(double) :	log	exp
Trajanje(ns)	161	370	24	154	1140	1800

Tablica 6.3.1 Vremena izvoðenja nekih matematièkih operacija

Zanimljivo je primjetiti da je množenje realnih brojeva višestruko brže od realnog dijeljenja, koje je isto kao i cjelobrojno množenje, dok je cjelobrojno dijeljenje još dvaput sporije. Složenije funkcije, logaritamska i eksponencijalna, su za red do dva sporije.

Na slici 6.3.1 su prikazana vremena stvaranja, sinkronizacije i matematièkih operacija, i to onoliki broj svake operacije da bi bilo približno istog reda.



Slika 6.3.1 Usporedba vremena stvaranja, sinkronizacije i matematièkih operacija

Iz slike se može dobiti uvid o korisnosti upotrebe višedretvenosti za razlièite probleme, uspoređujuæi potreban broj stvaranja dretvi, sinkronizacije i raèunanja.

7. UPOTREBA VIŠEDRETVENOSTI NA SUSTAV ZADATAKA

7.1 Pojmovi i definicije

Zadatak **T** je cjelovita jedinica aktivnosti, određen svojim vanjskim efektima: ulaznim podacima, rezultatima i trajanjem izvođenja. Uz svaki zadatak su pridružena dva događaja: poèetak i kraj. Zadaci se izvode u raèunalnom sustavu te mijenjaju stanje sustava između ta dva događaja.

Neka su:

- $f_1, f_2, ..., f_N$ instrukcije zadatka **T**
- $\underline{\mathbf{r}}$ skup registara
- $\underline{\mathbf{m}}$ skup memorijskih lokacija.

Stanje sustava se tada može prikazati sa:

$$\underline{\mathbf{s}} = [\ \underline{\mathbf{r}}\ \underline{\mathbf{m}}\ ].$$

Izvođenjem zadataka sustav poprima stanja $\underline{\mathbf{s}}_0, \underline{\mathbf{s}}_1, ..., \underline{\mathbf{s}}_N$, od kojih je $\underline{\mathbf{s}}_0$ poèetno, a $\underline{\mathbf{s}}_N$ završno stanje. Stanje $\underline{\mathbf{s}}_{k+1}$ dobiva se iz stanja $\underline{\mathbf{s}}_k$ obavljanjem instrukcije f_k , odnosno:

$$\underline{\mathbf{s}}_{k+1} = f_k\ (\ \underline{\mathbf{s}}_k).$$

Skup memorijskih lokacija zadataka možemo podijeliti na:

- domenu $\underline{\mathbf{D}} \in \underline{\mathbf{m}}$
- kodomenu $\underline{\mathbf{R}} \in \underline{\mathbf{m}}$.

Za dva zadatka $\mathbf{T}_i$ i $\mathbf{T}_j$ se kaže da su *nezavisni* ako su ispunjena slijedeæa tri uvjeta:

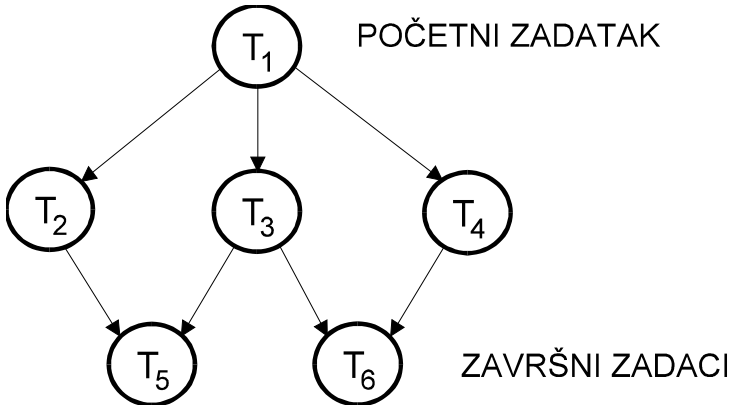
- $\underline{\mathbf{D}}_i \cap \underline{\mathbf{R}}_j = \varnothing$
- $\underline{\mathbf{D}}_j \cap \underline{\mathbf{R}}_i = \varnothing$
- $\underline{\mathbf{R}}_i \cap \underline{\mathbf{R}}_j = \varnothing$,

odnosno, takvi se zadaci mogu izvoditi neovisno jedan o drugom, pa i istovremeno. Ako bilo koji od uvjeta nije ispunjen, ako jedan zadatak koristi podatke koji drugi mijenja, tada su zadaci *zavisni* te se mora propisati redoslijed njihova obavljanja:

$$\mathbf{T}_i < \mathbf{T}_j \text{ ili } \mathbf{T}_j < \mathbf{T}_i.$$

Sustav zadataka je uređeni skup zadataka $\mathbf{T_1}, \mathbf{T_2}, ..., \mathbf{T_N}$, od kojih su svaka dva zadatka $\mathbf{T}_i$ i $\mathbf{T}_j$ nezavisna, ili vrijedi $\mathbf{T}_i < \mathbf{T}_j$ ili $\mathbf{T}_j < \mathbf{T}_i$.

Sustav zadataka se može prikazati grafom kao na slici 7.1.



Slika 7.1 Sustav zadataka prikazan grafom

U ovom poglavlju je razmatrana upotreba višedretvenosti na riješavanje sustava zadataka. U tu je svrhu napravljen i program za simulaciju izvoðenja sustava zadataka.

7.2 Simuliranje izvoðenja sustava zadataka

Program za simulaciju izvoðenja sustava zadataka sastoji se od tri dijela. Prvi dio izvodi zadatke jedan za drugim, prema uređenju. To je najjednostavniji naèin rasporeda zadataka koji potpuno zadovoljava u jednoprocesorskim sustavima.

Drugi dio stvara za svaki zadatak po jednu dretvu. Sinkronizacija se obavlja uz pomoæ uvjetnih varijabli. Svaki zadatak sadrži podatke o tome koliko ima neposrednih prethodnika te koliko i koje neposredne slijedbenike. Radi jednostavnosti i bez smanjenja opæenitosti pretpostavljeno je da zadaci sa manjim brojem ne mogu biti slijedbenici zadataka sa veæim brojem u prvom obavljanju sustava zadataka. Navedeni podaci i pravilo određuju uređenost sustava i dovoljni su za ispravno voðenje, tj. sinkronizaciju.

Treæi dio stvara onoliko dretvi koliko ima procesora, odnosno, koliko mu korisnik zada. Te dretve, zatim, obavljaju zadatke poštujuæi uređenje sustava. To, zapravo predstavlja idealan sluèaj, koji bi trebao rezultirati najkraæim vremenom izvoðenja. Kako je to u stvarnosti ovisi o samom sustavu zadataka (trajanja, uređenost), ali i također o optereæenosti sustava na kome se simulacija obavlja od strane ostalih procesa (korisnika).

Kada je sustav optereæen tada do izražaja dolazi vrijeme koje sustav posveæuje programu. Pošto jezgra sustava rukuje sa LWP-om kao osnovnom jedinicom, te pošto se radi o višekorisnièkom sustavu, jezgra æe programu dodijeliti onoliko svog vremena koliki je udio njegovih LWP-ova u cijelom sustavu. U takvim sluèajevima prije æe završiti oni višedretveni programi koji imaju više dretvi (LWP-a), nego oni sa manje, a istim ukupnim poslom.

U sluèaju neoptereæenog sustava do izražaja dolazi vrijeme promjena konteksta procesora koje uzrokuje mnoštvo dretvi, naspram skupa dretvi èiji je broj jednak broju procesora, kada je to vrijeme efektivno minimalno.

Trajanje zadatka određeno je brojem matematièkih operacija nad realnim brojevima:

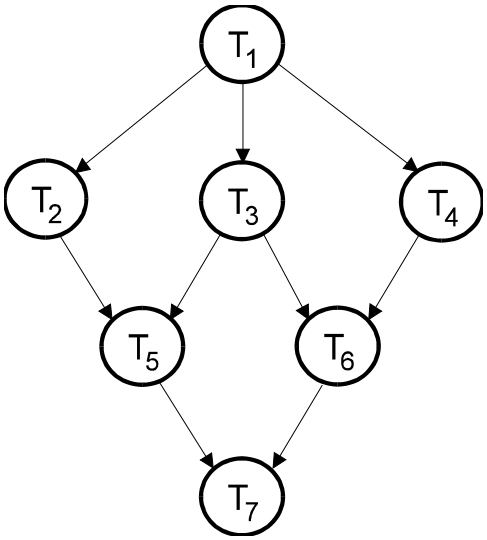
zbrajanjem/oduzimanjem, množenjem, dijeljenjem, logaritmiranjem i eksponenciranjem.

Zadaci iz stvarnog svijeta su veæinom takvi. Razlika jest obièno u operandima, odnosno, njihovim lokacijama, zbog kojih bi neke operacije trajale nešto dulje (kraæe). Upotrebljavaju se realni brojevi što možda bolje prezentira probleme koji se pojavljuju u praksi.

7.3 Sustav zadataka

Pod naslovom se podrazumjeva takav sustav koji se sastoji od skupa zadataka koji se jednom izvode. Zbog velike granulacije sistemskog sata (0.01 sekunda) trajanja zadataka sa manjim brojem operacija se ne mogu mjeriti (traju 0.00 sekundi). U ovom poglavlju su zato korišteni primjeri sustava sa nešto veæim brojem operacija, dok se u slijedeæem zadaci i sa manjim brojem operacija, ali sa ponavljanjem.

Prvi jest sustav od sedam zadataka prikazan na slici 7.3.1. Radi jednostavnosti svi zadaci traju jednako (isti broj matematièkih operacija).



Slika 7.3.1 Prvi primjer sustava

Sustav ima jedan poèetni i jedan završni zadatak koji se obavljaju slijedno. Zadaci između dopuštaju određenu paralelnost u izvođenju. Tako npr. kada završi prvi zadatak, zadaci 2, 3 i 4 se mogu izvoditi paralelno.

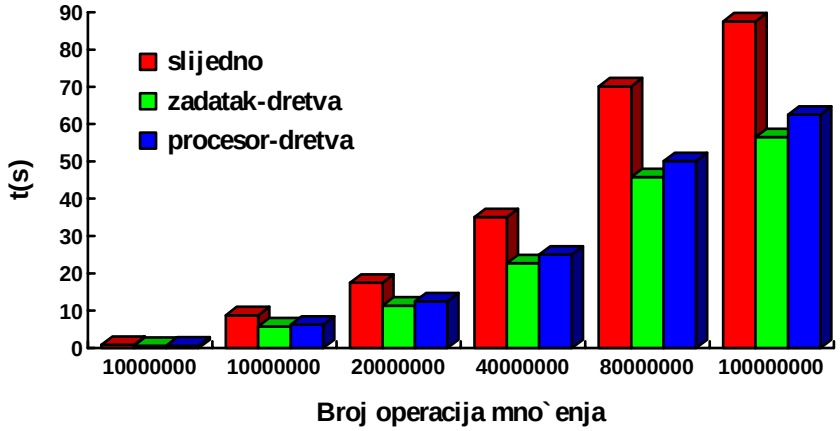
Rezultati, navedeni u tablici 7.3.1, pokazuju trajnja izvoðenja sustava zadataka u ovisnosti o broju matematièkih operacija.

N	T	t_{SR}	t_{SU}	t_{AR}	t_{AU}	t_{OR}	t_{OU}
10^6	0.125	0.875	0.88	0.582	0.88	0.626	0.89
10^7	1.25	8.76	8.76	5.76	8.77	6.26	8.78
2×10^7	2.5	17.50	17.51	11.33	17.54	12.50	17.54
4×10^7	5.01	35.07	35.08	22.70	35.05	25.0	35.05
8×10^7	10.02	70.12	70.13	45.76	70.09	50.05	70.11
10^8	12.5	87.55	87.55	56.54	87.64	62.52	87.58

Tablica 7.3.1 Rezultati simulacije

LEGENDA:
N - broj operacija množenja
T - prosjeèno trajanje zadatka
t_{XY} - proteklo vrijeme koje određuju indeksi X i Y
X: S - slijedno izvoðenje zadataka
A - svaki zadatak izvodi zasebna dretva
O - ima onoliko dretvi koliko i procesora (optimalno)
Y: R - trajanje u stvarnom (realnom) vremenu
U - trajanje u korisnièkom vremenu

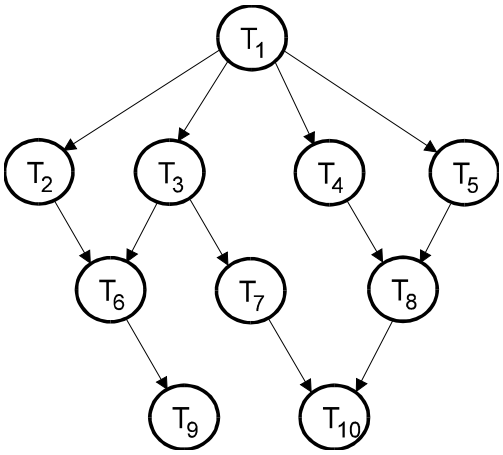
Iz tablice se vidi da je ukupno korisnièko vrijeme približno isto za sva tri naèina izvoðenja sustava zadataka. Stvarno vrijeme je kraæe za paralelne metode od slijedne što se vidi i iz slike 7.3.2.



Slika 7.3.2 Trajanje izvoðenja sustava zadataka

Ubrzanje koje se postiže je ogranièeno međusobnom ovisnosti zadataka. Ipak, ono je prilièno veliko, oko 1.5, od maksimalno moguæeg 2 (dva procesora sustava), što pokazuje da je opravadana upotreba višedretvenosti na ovakve sustava zadataka, koji imaju određene moguænost paralelizacije, a ne traju vrlo kratko.

Drugi primjer sastoji se od 10 zadataka prikazanih na slici 7.3.3.



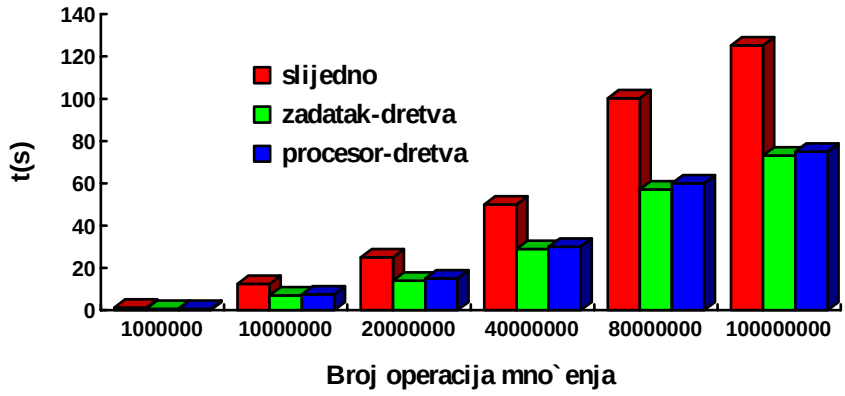
Slika 7.3.3 Drugi primjer sustava

Prikazani sustav ima donekle manje ovisnosti među zadacima tako da se dadu bolje paralelizirati. Rezultati simulacije, prikazani u tablici 7.3.2, to i potvrđuje.

N	T	t_{SR}	t_{SU}	t_{AR}	t_{AU}	t_{OR}	t_{OU}
10^6	0.125	1.25	1.25	0.72	1.26	0.75	1.25
10^7	1.25	12.53	12.53	7.01	12.53	7.52	12.52
2×10^7	2.5	25.05	25.05	13.98	25.07	15.01	25.04
4×10^7	5.01	50.08	50.05	29.01	50.09	30.04	50.11
8×10^7	10.02	100.2	100.2	57.08	100.2	60.03	100.2
10^8	12.5	125.2	125.2	73.2	125.3	75.0	125.2

Tablica 7.3.2 Rezultati simulacije

Rezultati prikazani u tablici pokazuju da je višedretveno, paralelno izvođenje mnogo kraæe od slijednog. To se jasno vidi iz slike 7.3.4.



Slika 7.3.4 Trajanje izvođenja simulacije

Ubrzanje iznosi prosjeèno 1.8, što je veæe nego li u prethodnom primjeru.

7.4 Sustav zadataka sa ponavljanjem

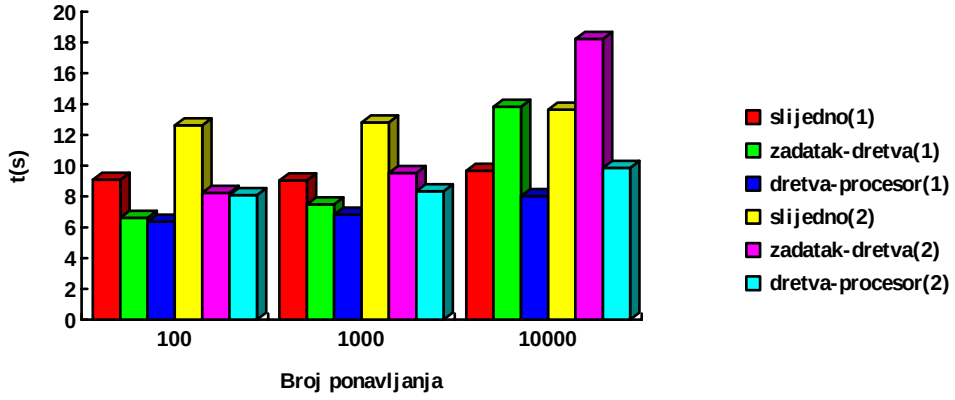
Sustavi zadataka iz prethodnog pglavlja su modificirani tako da se sustav obavlja mnogo puta, tj. obavlja se iteracija. Dodane su zavisnosti između zadnjih zadataka i prvog zadatka. Broj operacija po zadatku je adekvatno smanjen, tako da je ukupan broj operacija po zadatku, raèunajuæi sve iteracije, isti, odnosno, ukupno 10^7 množenja.

U tablici 7.4.1, te na slici 7.4.1, se nalaze stvarna vremena izvoðenja prvog i drugog sustava zadataka sa razlièitim brojem ponavljanja, ali istim ukupnim brojem operacija množenja.

	sustav sedam zadataka			sustav deset zadataka		
#ponavljnja	$t_{slijedno}$	$t_{zadatak\text{-}dretva}$	$t_{dretva\text{-}procesor}$	$t_{slijedno}$	$t_{zadatak\text{-}dretva}$	$t_{dretva\text{-}procesor}$
100	9.09	6.62	6.36	12.61	8.23	8.09
1000	9.04	7.48	6.82	12.80	9.53	8.33
10000	9.67	13.81	8.03	13.63	18.22	9.85

Tablica 7.4.1 Vremena izvoðenja sustava sa ponavljanjima

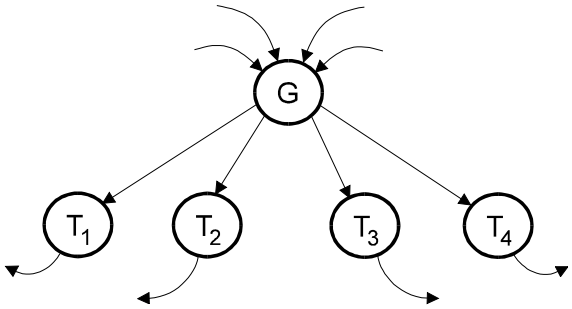
Ubrzanje ovisi o broju iteracija. Pri manjem broju iteracija, odnosno pri veæim zadacima, ubrzanje je približno isto za obe paralelne metode (oko 1.4). Pri veæem broju iteracija najbrža je metoda *procesor-dretva*. Pri 10000 iteracija metoda *dretva-procesor* jest i dalje najbolji, ali je metoda *zadatak-dretva* najsporija, èak i preko 30% od slijedne, a to uzrokuje veliko vrijeme sinkronizacije među mnoštvom dretvi.



Slika 7.4.1 Vremena izvoðenja sustava sa ponavljanjima

Slijedno vrijeme, također raste sa poveæanjem broja ponavljanja, a to uzrokuje nesavršenost samog programa za simulaciju, jer poveæanjem broja ponavljanja, odnosno, smanjenjem broja operacija po jednoj iteraciji, vrijeme izvoðenja zadatka postaje mjerljivo sa vremenom pozivanja funkcija za obavljanje tih operacija, kao i drugih potrebnih akcija.

Slijedeæi primjer simulacije jest *problem trgovaèkog putnika*[4]paralelnom metodom. Sustav zadataka kojim se simuliraju iteracije TSP-a[5]sastoji se od više zadataka. Jedan zadatak predstavlja onaj zadatak koji određuje poèetak iteriranja, èeka na kraj jedne razine iteriranja te obavlja eventualnu promjenu konfiguracije. Slijedeæi zadaci (2,4,8) predstavljaju broj iteracija koje se paralelno izraèunavaju. Sustav sa èetiri takva zadatka prikazan je na slici 7.4.2.



Slika 7.4.2 Sustav zadataka za TSP

Rezultati simuliranja su prikazani u tablici 7.4.2.

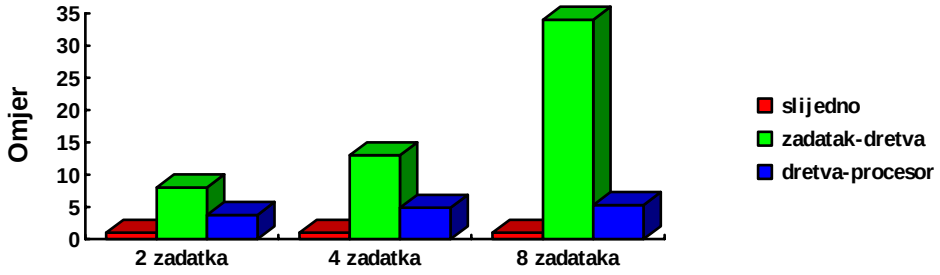
Broj paralelnih iteracija	$t_{slijedno}$	$t_{zadatak\text{-}dretva}$	$t_{dretva\text{-}procesor}$
2	3.85	30.9	14.5
4		50.7	18.7
8		129.5	20.4

Tablica 7.4.2 Vremena simuliranja TSP-a

Za slijedno raèunanje nema smisla upotrijebljavati više od jednog zadatka raèunanja, pa je navedeno vrijeme za takav sluèaj. I to je vrijeme preveliko zbog ipak dva zadatka (sinkronizacija+raèunanje) te zbog nesavršenosti sustava za simuliranje.

Iz rezultata je oèito da se gotovo svo vrijeme troši na međusobnu sinkronizaciju. Što ima više dretvi (vezanih) to je i sinkronizacija sporija te tako je i metoda *zadatak-dretva* toliko puta sporija od metode *dretva-procesor.*

Produljnje trajanja je prikazano na slici 7.4.3.



Slika 7.4.3 Produljenje trajanja višedretvenošæu

[4]Opširnije o problemu trgovaèkog putnika u poglavlju 8.4
[5]*traveling salesman problem* - problem trgovaèkog putnika

Ovakva upotreba višedretvenosti na sustav zadataka jest zapravo ekstreman sluèaj iteriranja sustava zadataka sa vrlo velikim brojem iteracija i sa malim brojem matematièkih operacija, koji pokazuje da se ovakvi problemi mnogo bolje (brže) riješavaju na standardni, slijedni naèin.

8. PRIMJERI UPOTREBE VIŠEDRETVENOSTI

8.1 Množenje matrica

Množenje matrica je raèunski zahtjevan algoritam sa ukupno $N \times M \times P$ operacija množenja, gdje su *M, N* dimenzije prve matrice **A** te *N, P* druge **B**. Množenje matrica jest najbolji predstavnik algoritama koji se dadu lako i efikasno paralelizirati. Paralelizacija se može izvesti na više naèina, a to može ovisiti i o organizaciji sustava, odnosno, njegovih procesora.

U ovom sluèaju višekorisnièkog sustava, upotrijebljena je metoda kod koje se paralelno raèunaju retci matrice umnoška. Svaka dretva, pomoæu jedne zajedniène varijable, utvrðuje slijedeæi redak izlazne matrice koji treba izraèunati, poveæava tu varijablu te raèuna utvrðeni redak. Navedeni kod se zaštiæuje zakljuèavanjem. Kada završi sa raèunanjem retka, uzima slijedeæi neizraèunati, sve dok takav postoji. Na ovaj naèin se smanjuje i onako mala meðusobna zavisnost između dretvi, koje sustav može na razlièite naèine rasporeðivati na svoje procesore, ukoliko ih ima manje nego dretvi.

Algoritam za paralelno množenje matrica **A** i **B** dimenzija $N \times M$ i $M \times P$ izgleda ovako:

```
procedura dretva;
    zaključaj;
    dok je redak<N radi
        mojred:=redak;
        redak:=redak+1;
        otključaj;
        za i:=1 do P radi
            za j:=1 do N radi
                C[mojredak, i]:= C[mojredak, i]+ A[mojredak, j]×B[j, i];
        zaključaj;
    otključaj;
kraj dretve.
```

Poèetna dretva najprije stvori ostale, koje odmah poènju izvoditi navedeni algoritam, te se onda i ona sama pridružuje raèunaju, da bi po završetku prièekala kraj svih ostalih dretvi.

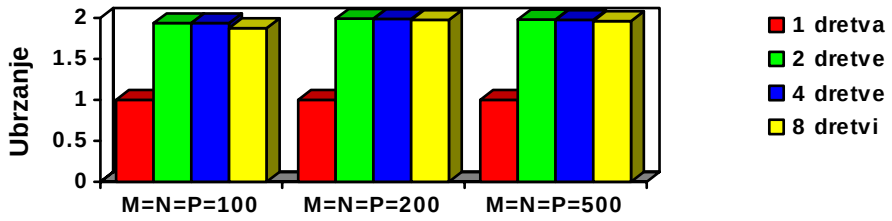
Rezultati, navedeni u tablici 8.1.1 te prikazani grafièki na slici 8.1.1, pokazuju da je ubrzanje proporcionalno broju procesora sustava (ukoliko ima barem isto toliko dretvi), odnosno, u ovom konkretnom sluèaju ubrzanje jest dvostruko. Ukupno potrošeno vrijeme sustava je gotovo isto za bilo koji (razuman) broj dretvi, dok je stvarno (realno) vrijeme dvaput kraæe pri paralelnom spram slijednog izraèunavanja.

Zbog mjerenja navedeni su primjeri množenja nešto veæih matrica, odnosno, dimenzija 100×100, 200×200 te 500×500. Zbog vrlo male ovisnosti među dretvama, odnosno, trenutka zakljuèavanja radi određivanja slijedeæeg retka raèunanja, vremena sinkronizacije su vrlo mala te su zanemarena.

	M=*N*=*P*=100		*M*=*N*=*P*=200		*M*=*N*=*P*=500	
#dretvi	t_{REAL}(s)	$t_{USER+SYS}$(s)	t_{REAL}(s)	$t_{USER+SYS}$(s)	t_{REAL}(s)	$t_{USER+SYS}$(s)
1	0.12	0.11	0.995	0.99	40.14	40.06
2	0.062	0.12	0.500	0.99	20.27	40.41
4	0.062	0.12	0.501	1.00	20.34	40.65
8	0.064	0.14	0.504	1.01	20.47	40.87

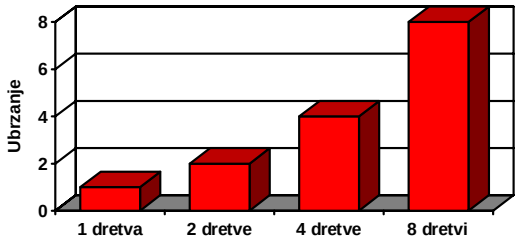
Tablica 8.1.1 Trajanja raèunanja množenja matrica

Izvoðenje programa je izvedeno na neoptereæenom sustavu, na kojem se najbrže množenje obavlja kada radi onoliki broj dretvi (vezanih dretvi) koliko ima i procesora, što je u ovom sluèaju dva. Ipak, razlika između trajanja programa sa dvije dretve i èetiri, odnosno, osam jest neznatna, jer množenje jednako napreduje uz bilo koje dvije trenutno aktivne dretve.



Slika 8.1.1 Ubrzanje u ovisnosti o broju dretvi

Iz rezultata se jasno vidi da ubrzanje ne ovisi o broju dretvi, ukoliko ih ima više nego li procesora sustava. Kada bi sustav imao èetiri, odnosno, osam procesora, ubrzanje bi bilo isto toliko, tj. èetverostruko ili osmostruko. Ustvari, bilo bi nešto manje (decimale) zbog ipak postojeæe male ovisnosti među dretvama. Na slici 8.1.2 je prikazano takvo hipotetsko ubrzanje za sluèaj osam procesora i različitog broja dretvi.



Slika 8.1.2 Hipotetsko ubrzanje

Množenje matrica, dakle, spada u grupu algoritama kod kojih je opravdano i preporuèeno korištenje višedretvenosti radi ubrzanja, ali samo tamo gdje dimenzije matrica nisu male. Ako su dimenzije male tada do izražaja dolazi vrijeme stavaranja dretvi i sinkronizacije te se ne isplati njihovo korištenje. Kvantitativno, to znaèi da broj operacija množenja mora biti barem toliki da traje za red više nego li stvaranje dretve.

8.2 LU dekompozicija

Poznata metoda riješavanja sustava linearnih jednadžbi jest metoda ***LU dekompozicije***.

Neka je:

- **A** - matrica koeficijenata uz nepoznanice
- **x** - vektor nepoznanica
- **b** - vektor slobodnih koeficijenata,

tada se sustav može prikazati u matrièoj formi:

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}.$$

Metoda *LU dekompozicije* se, najkraæe reèeno, sastoji u tome da se matrica **A** rastavi na dvije trokutne matrice:

- donju trokutnu **L** i
- gornju trokutnu **U**,

tako da vrijedi: $\mathbf{A} = \mathbf{L}\,\mathbf{U}$.

Vektor riješenja, tj. nepoznanica se tada dobije u dva koraka riješavanjem dva sustava:

$$\mathbf{L}\,\mathbf{y} = \mathbf{b}$$
$$\mathbf{U}\,\mathbf{x} = \mathbf{y}$$

koji se lako riješavaju supstitucijama prema naprijed i natrag, pošto su matrice trokutne.

Dekompozicija se može obaviti koristeæi matricu **A** za smještaj matrica **L** i **U**, tako da donja trokutna matrica **L** saèinjava donji trokut matrice **A** bez dijagonale (matrica **L** ima jedinice na dijagonali), a gornja trokutna **U** gornji trokut matrice **A**.

Algoritam paralelne dekompozicije za podreðene dretve u osnovi izgleda ovako:

```
procedura LU_dekompozicija(A,N);
    zakljuèaj;
    dok je i<N radi
        èekaj_start;
        dok je j≤N radi
            moj_j=j;
            j++;
            otkljuèaj;
            A[moj_j,i]:=A[moj_j,i]/A[i,i];
            za k:=i+1 do N radi
                A[j,k]:=A[moj_j,k]-A[moj_j,i]*A[i,k];
            zakljuèaj;
        signaliziraj_kraj;
    otkljuèaj;
kraj.
```

Primjenjeni algoritam je nešto veæi od navedenog, a razlike su u pivotiranju po retcima te provjeravanje vodeæeg elementa na dijagonali da li je nula, kada se on postavlja na jedinicu te se to pamti za daljnji postupak (*Householder*-ov postupak). Također, funkcija glavne dretve je nešto drukèija.

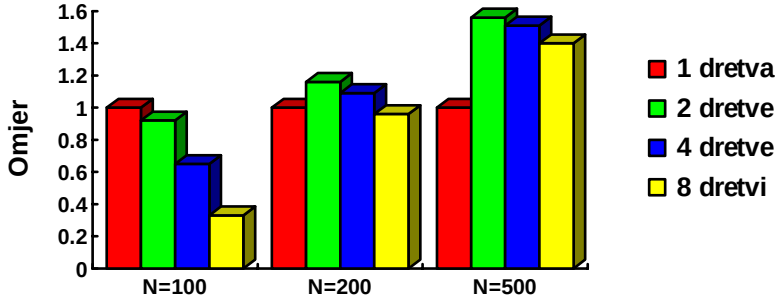
Pošto su iteracije druge petlje (iteracije po 'j') međusobno nezavisne, one se mogu paralelizirati pomoæu višedretvenosti. Dretve tako raèunaju po jednu iteraciju u jednom prolazu, odnosno, jedan redak trenutne podmatrice, sve dok nisu sve izraèunate. Kada je raèunanje podmatrice završeno, tada jedna dretva (glavna) utvrđuje najveæi element u prvom stupcu slijedeæe, za red manje, podmatrice (i+1, i+1 $\Rightarrow$ N, N), obavi pivotiranje te signalizira poèetak raèunanja podmatrice. Ta ista dretva mora, na kraju raèunanja podmatrice, prièekati kraj raèunanja svih ostalih dretvi.

Rezultati dobiveni izvođenjem programa prikazani su u tablici 8.2.1.

N=100				
#dretvi	t_{REAL}(s)	t_{USER}(s)	t_{SYS}(s)	$t_{USER+SYS}$(s)
1	0.053	0.05	0	0.05
2	0.058	0.07	0.01	0.08
4	0.082	0.1	0.06	0.16
8	0.162	0.09	0.18	0.27
N=200				
#dretvi	t_{REAL}(s)	t_{USER}(s)	t_{SYS}(s)	$t_{USER+SYS}$(s)
1	0.387	0.38	0	0.38
2	0.333	0.57	0.01	0.59
4	0.354	0.62	0.07	0.69
8	0.402	0.66	0.14	0.8
N=500				
#dretvi	t_{REAL}(s)	t_{USER}(s)	t_{SYS}(s)	$t_{USER+SYS}$(s)
1	6.504	6.47	0	6.47
2	4.181	7.89	0.08	7.97
4	4.321	8.2	0.25	8.45
8	4.642	8.64	0.41	9.05

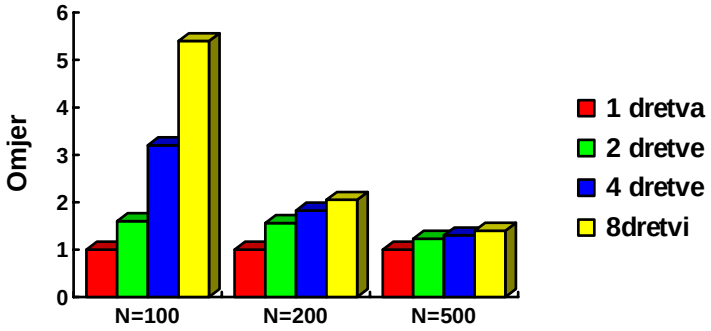
Tablica 8.2.1 Trajanja LU dekompozicija

Rezultati, također prikazani na slici 8.2.1, pokazuju da se ubrzanje postiže tek pri dekompoziciji matrice dimenzije veæe od 200×200, odnosno, za manje matrice vrijeme sinkronizacije je toliko da usporava raèunanje i u odnosu na slijednu metodu. Pri dekompoziciji matrice dimenzija 500×500 ubrzanje jest osjetno, tj. prosjeèno 1.56 za program sa dvije dretve.



Slika 8.2.1 Ubrzanje LU dekompozije višedretvenošæu

Ukupno potrošeno vrijeme sustava, prikazano na slici 8.2.2, pokazuje znaèajan rast sa porastom broja dretvi, a uzrokuje ju poveæanje komunikacije.



Slika 8.2.2 Omjer ukupnog vremena sustava

Upotreba višedretvenosti pri LU dekompoziciji opravdana je jedino u sluèaju matrica veæih dimenzija. Ubrzanje je to veæe što je matrica veæa te što sustav ima više procesora.

8.3 Sortiranje

Sortiranje i pretraživanje velikih polja je raèunski, odnosno, vremenski zahtjevan problem. Pokušaj paralelizacije jest stoga opravdan, te je u ovom poglavlju prikazan primjer paralelnog višedretvenog sortiranja metodom *quick sort*.

Quick sort jest poznati brzi rekurzivni algoritam sortiranja koji radi tako da polje dijeli (sortira) na dva dijela, èiji su elementi manji, odnosno, veæi od nekog središnjeg elementa. Svaki se takav dio dalje rekurzivno dijeli na dva dijela, sve dok ne ostanu dva elementa u dijelu.

Paralelizacija se sastoji u tome da, kada dretva obavi podjelu polja na dva dijela, jedan dio obavlja sama dretva dok za drugi stvori novu. Tako svaki put nakon podjele, ukoliko je dio dovoljno velik da vrijeme stavranja dretve ne bude veæe od vremena samog sortiranja. Razlika između navedenog algoritma i predloženog potprograma jest u tome što se funkcija ne poziva rekurzivno, veæ je to nadomješteno simuliranjem stoga, na koji se postavaljaju bitne informacije o dijelu koji treba sortirati, a u svrhu ubrzanja algoritma.

Potprogram prima iste parametre kao i standardna funkcija *qsort*, što je èini upotrebljivom na sve tipove podataka. Potprogram je napisan tako da se može ogranièiti broj istovremeno aktivnih dretvi pomoæu jedne globalne varijable.

Osnovni algoritam paralelnog sortiranja izgleda ovako:

```
procedura qsort(A,N);
    ako je N<3 tada
        ako je A[1]>A[2] tada
            zamijeni A[1] i A[2];
    inaèe
        prvi:=1;
        zadnji:=N;
        srednji:=(prvi+zadnji)/2;
        dok je prvi<zadnji-1 radi
            dok je A[prvi]<A[srednji] ∧ prvi<srednji radi
                prvi:=prvi+1;
            ako je prvi==srednji tada
                srednji:=(prvi+zadnji)/2;
            dok je A[zadnji]>A[srednji] ∧ zadnji>srednji radi
                zadnji:=zadnji-1;
            ako je zadnji==srednji tada
                srednji:=(prvi+zadnji)/2;
            zamijeni A[prvi] i A[zadnji];
        ako je srednji>MINIMUM tada
            stvori_dretvu(qsort, A, rednji);
        inaèe
            pozovi qsort(A, srednji);
        pozovi qsort((adresa)A[srednji+1], N-srednji);
kraj.
```

Potprogram je isproban na polju niza znakova nekoliko razlièitih duljina i razlièitog broja elemenata polja. Rezultati su prikazani u tablici 8.3.1.

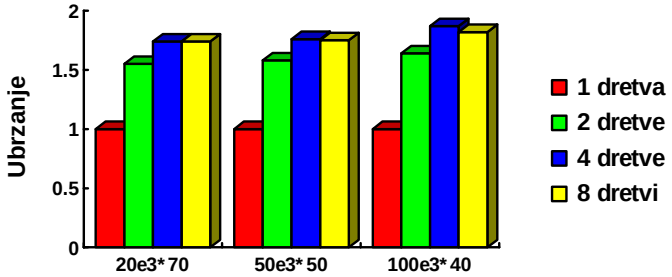
	20e3*70 (s)	50e3*50 (s)	100e3*40 (s)
1 dretva	0.546	1.28	2.5
2 dretve	0.352	0.808	1.52
4 dretve	0.313	0.725	1.34
8 dretvi	0.313	0.733	1.37

Tablica 8.3.1 Trajanja sortiranja

Oznake 20e3*70, 50e3*50 i 100e3*40 predstavljaju broj elemenata polja (20e3, 50e3 i 100e3) te velièinu jednog elementa u znakovima (70, 50, 40).

Ubrzanje prikazano na slici 8.3.1 jest ujednaèeno za sluèajeve do najviše èetiri i osam dretvi, dok je za sluèaj samo dvije dretve nešto manje. Ubrzanje ogranièava i sam algoritam. Naime, prvu podjelu obavlja samo jedna dretva, a to je ipak prilièan dio

sortiranja. Ipak, ubrzanje jest znaèajno te to pokazuje da se višedretvenost može efikasno upotrijebiti i u ovom podruèju.



Slika 8.3.1 Ubrzanje dobiveno višedretvenošæu

8.4 Problem trgovaèkog putnika

Problem trgovaèkog putnika je najpoznatiji problem kombinatorièke optimizacije. Problem je zadan sa skupom određenog broja gradova (N) i njihovom međusobnom udaljenošæu. Trgovaèki putnik treba obiæi sve gradove i vratiti se u polazni grad, ali tako da put, tj. duljina pređenog puta bude minimalna. Ukupno ima $\frac{(N-1)!}{2}$ razlièitih moguæih puteva, što je prepreka egzaktnom pronalaženju minimuma, tj. problem jest NP težak. Zbog toga se problem riješava aproksimacijskim metodama od kojih je najpoznatija metoda ***simuliranog kaljenja*** [3], koja je ime dobila po istoimenom postupku koji je razvio **N. Metropolis**.

Problem kombinatorièke optimizacije može se formalizirati kao uređeni par ($\mathcal{R}$, C), gdje je $\mathcal{R}$ skup moguæih konfiguracija i C funkcija cilja, odnosno u konkretnom sluèaju to je duljina puta. Metoda *simuliranog kaljenja* sastoji se u tome da se iz poèetne konfiguracije (odabrane na sluèajan naèin), preko mnogo malih promjena, doðe do konfiguracije minimalne funkcije cilja (duljine puta). U svakoj takvoj iteraciji na sluèajan naèin se generira određena promjena konfiguracije, tj. prijelaz u susjednu konfiguraciju (zamjena obilaska puta između dva grada). Ako nastala promjena uzrokuje smanjenje funkcije cilja, tada se ona prihvaæa. Ukoliko ona uzrokuje poveæanje, promjena se prihvaæa ako je ispunjen uvjet:

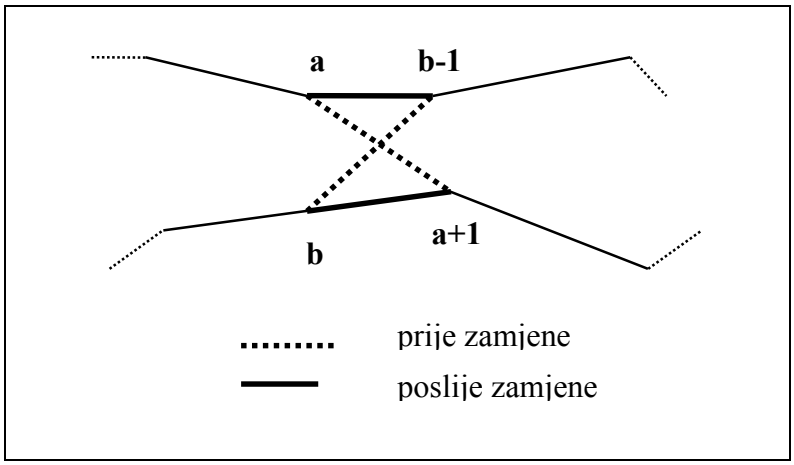
$$\exp(-\Delta C/T) > \text{random}[0,1)$$

gdje T predstavlja ekvivalent temperature za određeni problem. Iteriranje se sprovodi u koracima u kojima je temperatura konstantna, a među kojima se smanjuje. Prihvaæanje nekih prijelaza koji uzrokuju poveæanje puta omoguæuje bijeg iz lokalnog minimuma. Funkcija cilja konaène konfiguracije veæinom nije optimalna, ali je vrlo blizu, što je dovoljno dobro riješenje u veæini sluèajeva.

Pa ipak i takva aproksimacijska metoda zahtijeva vrlo mnogo iteracija, tj. prilièno veliko vrijeme raèunanja. Radi smanjenja toga vremena, razvijene su razlièite metode paralelnog simuliranog kaljenja.

Metoda korištena u ovom primjeru jest metoda N-paralelnog simuliranog kaljenja prikazana u [2]. Metoda je izvorno primjenjena na multiprocesorskom sustavu (stotinjak procesora), dok je ovdje realizirana pomoæu dretvi. Ukratko, metoda u jednoj razini spekulativno raèuna više iteracija, pretpostavljajuæi da æe prethodne iteracije odbaciti generirane prijelaze. Metoda slijedi slijednu metodu tako da generira i prihvaæa sve prijelaze koje obavlja i slijedna metoda. Najviše iteracija u jednoj razini raèunanja prolazi se kada je u svakoj iteraciji, osim eventualno posljednje, odluka o prihvaæanju generiranog prijelaza negativna. Najmanje, odnosno, samo jednu, obavlja se kada prva iteracija prihvaæa prijelaz, kada se raèunanja svih ostalih zanemaruju (odbacuju).

Kod problema trgovaèkog putnika, prijelaz iz trenutne konfiguracije u susjednu (u primjenjenom algoritmu) obavlja se tako da se na sluèajan naèin odaberu dva grada te se raèuna promjena puta koja nastaje kada se promjeni redoslijed obilaska puta između ta dva grada, kako je prikazano na slici 8.4.1.



Slika 8.4.1 Promjena puta

Promjena duljine puta može se izraèunati i bez raèunanja ukupnog puta, koristeæi samo dio gdje dolazi do promjene, tj. promjena je jednaka:

$$\Delta d = d_{a,b-1} + d_{a+1,b} - d_{a,a+1} - d_{b-1,b}$$

gdje je $d_{i,j}$ udaljenost između gradova **i** i **j**. Radi jednostavnosti, gradovi su zadani koordinatama u ravnini, tako da međusobna udaljenost predstavlja udaljenost između dviju toèaka. Te su udaljenosti izraèunate za svaka dva grada i smještene u matricu $\underline{d}$, tako da je udaljenost između gradova **i** i **j** određena elementom $d_{i,j}$, odnosno, $d_{j,i}$. Put je predstavljen jednim poljem u kojem se nalaze kazaljke na određene gradove, tako da zamjena dva grada predstavlja zamjenu njihovih kazaljki (indeksa).

Rezultati su vrlo osjetljivi na ulazne parametre. Navedena su dva sluèaja: problem 100 te problem 500 gradova, uz parametre nevedene u tablici 8.4.1.

N	D_{MAX}	S	α	DML	P_0
100	1000	70	0.9	0.3333	0.8
500	1000	50	0.8	0.1	0.5

Tablica 8.4.1 Ulazni parametri

LEGENDA:
N - broj gradova
D_{MAX} - velièina kvadrata u kome su smješteni gradovi
S - broj koraka hlaðenja
α - faktor smanjenja temperature
DML - faktor za određivanje broja iteracija pri konstantnoj temperaturi
P_0 - vjerojatnost prihvaæanja prve *loše* konfiguracije

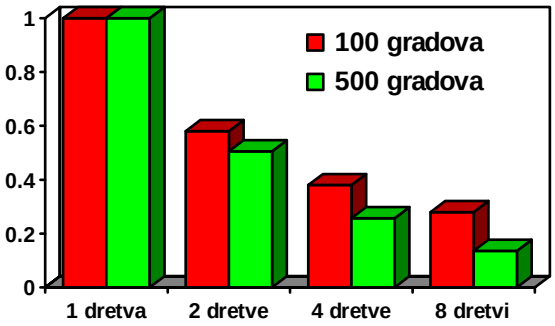
Dobiveni rezultati prikazani su u tablici 8.4.2.

100 gradova							
#dretvi	d_0	d_∞	#iteracija	#razina	t_{USER}	t_{SYS}	t_{REAL}
1	51776	8242	233310	233310	1.53	0.06	1.59
2	51776	8242	233310	135462	6.49	10.49	15.17
4	51776	8242	233310	88403	16.53	39.07	36.21
8	51776	8242	233310	66337	48	109	108
500 gradova							
#dretvi	d_0	d_∞	#iteracija	#razina	t_{USER}	t_{SYS}	t_{REAL}
1	254941	17972	1025000	1025000	8.5	0.16	8.92
2	254941	17972	1025000	518100	29.7	44.56	74.03
4	254941	17972	1025000	265109	58.5	108	125
8	254941	17972	1025000	139424	108	236	218

Tablica 8.4.2 Vremena izvoðenja

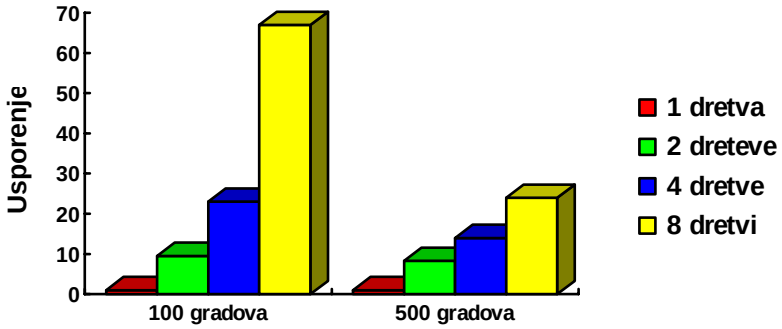
Iz tablice je vidljivo da broj razina raèunanja kod problema 100 gradova sve manje opada poveæavanjem broja dretvi, dok u drugom sluèaju to opadanje je gotovo proporcionalno broju dretvi. Objašnjenje se nalazi u ulaznim parametrima, tj. vjerojatnost prihvaæanja je mnogo manja pri problemu od 500 gradova (manji $\mathbf{P_0}$ i α). Kvaliteta je zato nešto lošija, ali je zato vrijeme raèunanja razumno (pri samo jednoj dretvi - slijedna metoda). Faktori opadanja broja razina raèunanja u ovisnosti o broju dretvi prikazani su na slici 8.4.2.

Slike pokazuju da bi paralelnim izvoðenjem program trebao biti brži od slijednog programa. Međutim, rezultati izvoðenja programa sa više dretvi traju višestruko dulje od slijednog programa, odnosno, programa sa samo jednom dretvom.



Slika 8.4.2 Opadanje broja razina raèunanja

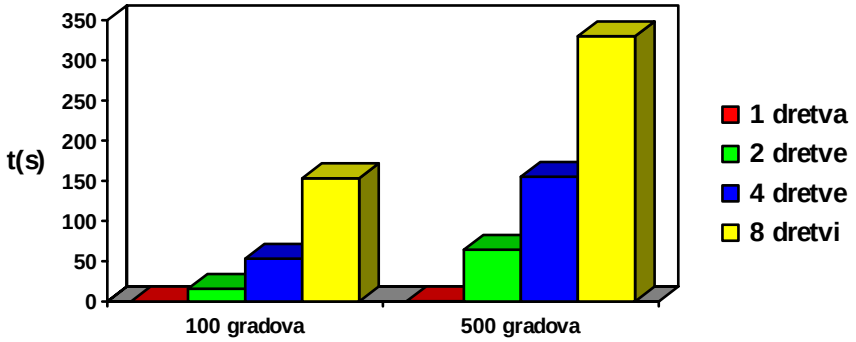
Rezultati, prikazani na slici 8.4.3, pokazuju da što je više dretvi to je trajanje raèunanja dulje. Na prvi pogled, rezultati su iznenađujuæi. Međutim, analizom rezultata vidi se da se više od 50% vremena troši na sustav, a također i korisnièko je vrijeme višestruko veæe od onog slijedne metode. Pošto ukupno trajanje samog raèunanja ne može biti znaèajnije veæe od onog slijedne metode, dolazi se do zaključèka da se ogromna veæina vremena troši na međusobnu sinkronizaciju.



Slika 8.4.3 Omjer trajanja ovisno o broju dretvi

Analizom programa vidi se da se raèunanje jedne iteracije svodi na dva generiranja sluèajnog broja, raèunanje promjene puta te odlukom o prihvatljivosti promjene, tj. jednom eksponencijelnom funkcijom, nekoliko množenja i dijeljenja te nešto više zbrajanja i oduzimanja. Vrijeme koje je za to potrebno jest vrlo kratko (manje od 10 μs). Raèunanju slijedi sinkronizacija, tj. glavna dretva mora prièekati da sve podređene dretve završe raèunanje te razine.

Pošto se radi o višekorisnièkom UNIX sustavu, komunikacija je bitno veæa od vremena raèunanja iteracije. Također, dretva po završenju iteracije èeka na uvjetnoj varijabli te joj sustav oduzima procesor dok se uvjet ne ispuni. To se oduzimanje dešava onoliko puta koliko ima razina iteriranja za svaku dretvu, što je vrlo velik broj koji pokazuje gdje se zapravo troši glavnina vremena. To je vrijeme promijene konteksta između dretvi nad procesorom, koje, inaèe, ne bi došlo do izražaja jer je vrlo malo, ali pomnoženo velikim brojem predstavlja znaèajno vrijeme.



Slika 8.4.4 Potrošeno vrijeme sustava

Na slici 8.4.4 je prikazano sistemsko vrijeme izvoðenja programa s obzirom na broj dretvi. Vrijeme samog raèunanja je izraèunato korištenjem slijedne metode te brojem razina raèunanja te je oduzeto od ukupnog korisnièkog i sistemskog vremena. Slika potvrðuje prethodne zaključke.

Paralelno simulirano kaljenje bi bilo brže od serijskog samo u onim višeprocesorskim sustavima u kojima bi vrijeme komunikacije bilo mjerljivo vremenu raèunanja iteracije (transputeri, ...), ili za one probleme za koje bi raèunanje jedne iteracije bilo mnogo dulje, tj. mjerljivo vremenu sinkronizacije. Problem trgovaèkog putnika, oèito nije takav problem te je njegova paralelizacija na ovakav naèin, na ovom sustavu, neupotrebljiva.

9. ZAKLJUÈAK

Mjerenje trajanja izvoðenja dretvi (stvarno, korisnièko i sistemsko vrijeme) razlikujemo u ovisnosti o tome je li dretvi stalno pridružen LWP (vezana dretva), ili pak više njih koristi iste (nevezane dretve). Uzrok razlici jeste to što dretve same nemaju moguænosti mjerenja tih vremena, veæ koriste moguænosti pripadnih LWP-ova. Ukoliko se radi o vezanoj dretvi, onda ona jednostavno koristi pripadni LWP, te je vrijeme dobiveno iz tog LWP-a vezano samo uz tu dretvu. Ukoliko dretva nije vezana i ukoliko nije jedina takva u tom procesu, tada se može desiti da se više njih obavljaju na istom LWP-u, te se ne može dobiti vrijeme koje je vezano samo uz tu dretvu, veæ samo ukupno vrijeme svih dretvi koje su koristile taj LWP. Moguænosti mjerenja koje nudi LWP jesu èetiri brojila vremena, koji odbrojavaju od određene zadane vrijednosti do nule. Granulacija vremena koja se može dobiti teoretski jesu mikrosekunde, ali je stvarna granulacija, osim stvarnog vremena, ovisna o granulaciji sistemskog sata, koji u promatranom sustavu iznosi stotinku sekunde. Upotreba orginalnih funkcija za rad sa tim brojilima je nešto kompleksnija te su napravljene jednostavnije koje vraæaju proteklo, a ne preostalo vrijeme.

Optereæenje sustava koje uzrokuje višedretveni program nije u direktnoj vezi sa brojem dretvi koje izvode program, veæ ono direktno ovisi o broju aktivnih LWP-ova programa. Pod pojmom ‘aktivni’ misli se na one LWP-e koji izvode određenu, tj. određene dretve koje konstantno nešto rade, a ne èekaju na određene događaje (zahtjeve korisnika). Ukoliko se program sastoji od vezanih dretvi, optereæanje koje uzrokuje program proporcionalno je broju takvih dretvi, te ukoliko je taj broj velik (>50), optereæenje može biti i tako veliko da znatno uspori sustav, te on postaje neupotrebljiv sa strane ostalih procesa, pa se takva upotreba treba izbjegavati.

Upotrebljivost višedretvenosti za ubrzanje nekih zadataka ovisi o kakvim je zadacima rijeè te na kakvom se sustavu radi. Ukoliko se radi o zadacima koji se dadu rastaviti na nezavisne ili malo zavisne dijelove te ukoliko ti dijelovi nisu trivijalni, tj. ako trajanje izvoðenja tih dijelova nije usporedivo sa vremenima stvaranja novih dretvi, odnosno, njihovom međusobnom sinkronizacijom, tada se korištenjem višedretvenosti postiže ubrzanje, uz ispunjenje osnovnog preduvjeta, a taj je da sustav ima barem dva procesora. Što je manja zavisnost među dijelovima, što je trajanje izvoðenja dijelova veæa te što sustav ima više procesora to je i ubrzanje veæe. Višedretvenost je tako primjenjiva na zadatke množenja matrica, razna sortiranja i pretraživanja te sliènih problema koji se dadu rastaviti na sustav zadataka sa malim međusobnim ovisnostima te dovoljno dugim trajanjem. Drukèiji problemi, tj. sustavi zadataka sa malim trajanjima te veæom međusobnom ovisnošæu, ne dadu se efikasno paralelizirati višedretvenošæu na ovakvim višekorisnièkim sustavima, jer komunikacija među dijelovima traje više od samog rada dretvi te njihova upotreba rezultira sporijem obavljanju od slijednog.

LITERATURA

[1] "**Solaris 2.4: Multithreaded Programing Guide**", *SunSoft, Sun Microsystems,* Mountain View, California, 1994.

[2] A. Sohn, "**Parallel N-ary Speculative Computation of Simulated Annealing**", *IEEE Transactions on Parallel and Distributed Systems,* vol. 6, str. 997-1005, Oct.1995.

[3] P.J.M. van Laarhoven, E.H.L. Aarts, "**Simulated annealing: Theory and Applications**", *D. Reidel Publishing Company,* Dordrecht, 1987.

[4] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, **"Solving problems on concurrent procesors: Volume I, General Techniques and Regular Problems"**, *California Institute of Technology, Prentice-Hall International,* Englewood Clifts, New Jersey, 1988.

[5] S. Turk, L. Budin, **"Analiza i projektiranje raèunalom"**, *Školska knjiga,* Zagreb, 1989.