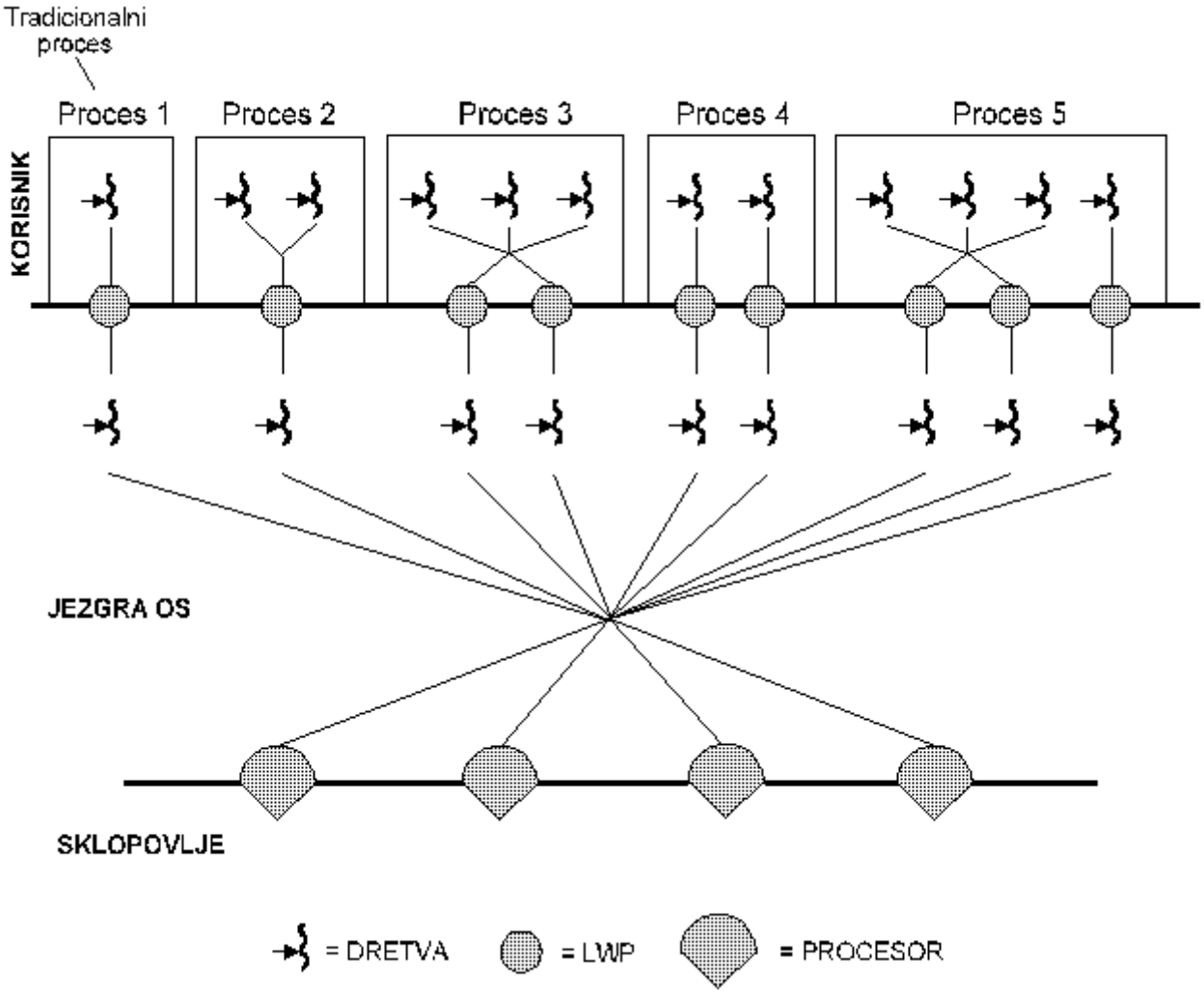
Seminar 2: VIŠEDRETVENOST

Povijest višedretvenog programiranja počinje 60-tih, dok se njihova implementacija na UNIX sustavima pojavljuje sredinom 80-tih godina, a na ostalim sustavima nešto kasnije. 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 izvođenja, već to obavlja sam operacijski sustav. Štoviše, ukoliko je to višeprocesorski sustav, onda se neke jedinice-dretve mogu izvoditi istovremeno. Komunikacija među dretvama je jednostavna i brža u odnosu na komunikaciju među procesima, jer se obavlja preko zajedničkog adresnog prostora, te se može obaviti 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 čija bi istovremena upotreba (i eventualno promjena) izazvala greške. Takvi se kritični odsječci programa zaštićuju međusobnim isključivanjem. 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 takoder zaštićuju zaključanjem i iz razloga što te funkcije zaključavanja/otključavanja također ažuriraju glavnu memoriju.

Arhitektura višedretvenog sustava Solaris 2.4



Operacijski sustav za koji su predviđene ove laboratorijske vježbe jest *Sun*-ov *Solaris 2.4* (ili noviji). Vidljivost dretvi jest samo unutar procesa, čiji su one dio i čija sredstva dijele (adresni prostor, otvorene datoteke, ...). Slijedeća stanja su, jedinstvena svakoj dretvi: *identifikacijski broj dretve, registarsko stanje uključujući programsko brojilo i kazaljku stoga, stog, signalna maska, priorite* i *privatni prostor same dretve.*

Upravljanje dretvi, odnosno, osiguravanje da se sve dretve izvode, obavlja se pomoću dretvene biblioteke *libthread*. Upravljanje se obavlja na korisničkoj razini, a ne na razini operacijskog sustava. Veza izmedu dretve i jezgre su tzv. *laki* (*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 predočiti jednostavno kao virtualni procesor.

Arhitektura višedretvenog sustava *Solaris 2.4* prikazana je na slici 1. Iz slike je vidljivo da standardni UNIX procesi imaju samo jedan LWP, odnosno, jednu dretvu upravljanja. Također, svaka dretva ne mora imati vlastiti LWP, već ih više njih može koristiti iste LWP-ove, odnosno, razlikujemo dvije skupine dretvi: *vezane* (engl. *bound threads*) i *nevezane* (engl. *unbound threads*). Prve, *vezane*, su takve dretve kojima je pridjeljen vlastiti LWP koji izvodi samo njene instrukcije, dok druge, *nevezane*, nisu vezane za vlastiti LWP, već se mogu izvoditi na bilo kojem LWP-u koji se nalazi na raspolaganju procesu, a nije vezan za jednu dretvu. Vezane dretve zauzimaju nešto više sredstava sustava, ali su zbog izbjegavanja učestalih promjena dretvi na LWP-u, brže.

Stvaranje dretvi

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

```
int thr_create( void *sp, size_t ss, void * (*početna_f) (void *),

void *arg, long zast, thread_t *nova_dr);
```

`sp` i `ss` su početna adresa i veličina stoga (uz `NULL`, tj. 0 uzimaju se standardne vrijednosti). `početna_f` je kazaljka na početnu, prvu funkciju čije instrukcije stvorena dretva počinje izvoditi. `arg` je kazaljka na argument koji se prenosi u stvorenu dretvu, ukoliko nije `NULL`. `zast` je skup bitova koji određuju karakteristike stvorene dretve, dok je `nova_dr` kazaljka na lokaciju gdje se sprema identifikacijski broj stvorene dretve.

Vrijednost `zast` obično se dobiva kao rezultat operacija logičkog zbrajanja između više različitih vrijednosti, kao što su `THR_BOUND` (stvara vezanu dretvu) i `THR_NEW_LWP` (ovom zastavicom povećavamo broj LWP-a za jedan).

Završetak rada dretve

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

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

`status` je kazaljka na stanje sa kojim dretva završava. Dretva čeka na završetak druge dretve pozivom funkcije `thr_join`.

```
int thr_join( thread_t čekana_dr, thread_t *dočekana_dr,

void **stanje);
```

`čekana_dr` je identifikacijski broj dretve na čiji se kraj čeka, ukoliko nije nula, kada se čeka na završetak bilo koje dretve. `dočekana_dr` je kazaljka na broj dretve koja je završila i dočekana je ovom dretvom, ukoliko nije `NULL`. `stanje` je kazaljka na kazaljku izlaznog statusa dočekane dretve. Funkcija `thr_join` zaustavlja izvođenje pozivajuće dretve sve dok određena dretva ne završi sa radom. Nakon ispravnog završetka funkcija vraća nulu.

Normalni završetak višedretvenog programa zbiva se kada sve dretve završe sa radom, odnosno, kada prva, početna dretva izađe 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`, ...).

Međusobna komunikacija i sinkronizacija

Promatrani operacijski sustav omogućuje četiri načina sinkronizacije među dretvama: *međusobno isključivanje, uvjetne varijable, zaključavanja čitaj/piši* i *semafori*. Kada je potrebno zaštititi neke dijelove programa od istovremenog korištenja više dretvi (kritični odsječci), tada se koriste funkcije međusobnog isključivanja, odnosno, dijelovi programa se zaključuju pomoću kontrolnih varijabli - ključeva. Inicijalizacija

ključeva se obavlja funkcijom `mutex_init`, zaključavanje funkcijom `mutex_lock`, a otključavanje funkcijom `mutex_unlock`.

```
int mutex_init(mutex_t *ključ, int tip, void *arg);

int mutex_lock(mutex_t *ključ);

int mutex_unlock(mutex_t *ključ);
```

`ključ` pokazuje na varijablu zaključavanja, `tip` određuje da li se sinkroniziraju dretve istog procesa (`USYNC_THREAD` ili i dretve različitih procesa(`USYNC_PROCESS`), dok se `arg` ne koristi. Sve dretve, koje pokušaju zaključati već zaključanu varijablu ostaju blokirane na pozivu sve dok varijabla ostaje zaključana. Kada se varijabla otključa, onda samo jedna dretva ulazi slijedeća u kritični osječak, uz zaključavanje varijable. Zaključavanjem se smanjuje moguća istovremenost u izvođenju skupine dretvi, te je njihova upotreba prihvatljiva (obavezna) samo u kritičnim odsječcima.

Uvjetne varijable se koriste kada želimo da neka dretva zaustavi svoje izvođenje te čeka 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_init`, `cond_wait`, `cond_signal` i `cond_broadcast`.

```
int cond_init(cond_t *uvjet, int tip, void *arg );

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

int cond_signal(cond_t *uvjet);

int cond_broadcast(cond_t *uvjet);
```

`uvjet` je kazaljka na uvjetnu varijablu, `ključ` jest kazaljka na varijablu zaključavanja, `tip` označuje tip sinkronizacije (kao i kod međusobnog isključivanja), dok se `arg` ne koristi. Pozivom `cond_wait` pozivajuća dretva se postavlja u stanje čekanja koje završava neka druga dretva pozivima `cond_signal` i `cond_broadcast`. Poziv `cond_signal` ispunjuje uvjet za nastavaka samo jedne dretve iz reda čekanja na istu uvjetnu varijablu, dok poziv `cond_broadcast` omogućuje svim takvim dretvama nastavak izvođenja. Ako niti jedna dretva nije bila blokirana na uvjetnoj varijabli prilikom poziva `cond_signal` i `cond_broadcast`, onda ovi pozivi nemaju nikakav učinak, tj. ako u slijedećem trenutku neka dretva pozove `cond_wait` ona ostaje blokirana. `cond_wait`< otključava `ključ`, ako ne prolazi, ali i ponovo zaključava kada prolazi.

Semafori se obično upotrebljavaju za pristup ograničenim sredstvima od strane većeg broja korisnika dretvi. Osnovne funkcije za rad sa semaforima u višedretvenom programu su: `sema_init`, `sema_post` i `sema_wait`.

```
int sema_init(sema_t *sem, unsigned int koliko, int tip, void *arg);

int sema_post(sema_t *sem);

int sema_wait(sema_t *sem);
```

`sem` je pokazivač na varijablu semafora, `koliko` je broj na koji se postavi semafor, `tip` isto kao i kod međusobnog isključivanja, dok se `arg` ne upotrebljava. Funkcija `sema_post` jedinično povećava semafor. Funkcija `sema_wait` smanjuje vrijednost semafora za jedan, ako je vrijednost semafora veća od nule, u protivnom čeka dok se vrijednost ne poveća.

Zadatak

1. Napisati program koji kreira zadani broj dretvi (D). Sve dretve trebaju obavljati isti posao:

```
Posao_dretve(N){
   za i=1 do N{
      A=A+1;
```

```
    }
}
```

Broj dretvi D i broj N se trebaju unijeti iz komandne linije (npr. program 2 10000) ili na početku izvođenja glavnog programa. Početna vrijednost varijable A je 1. Kad sve dretve obave svoj posao, treba ispisati vrijednost varijable A.

2. Proširiti program tako da se onemogući dretvama istovremeno pisanje i čitanje varijable A i to s pomoću MUTEX varijable zaključavanja.

Napomene

Prilikom prevođenja potrebno je postaviti zastavicu `-D_REENTRANT` koja ukazuje na to da se koriste višedretvene inačice upotrijebljenih funkcija, ako takve postoje, te zastavicu -*lthread* (npr. `cc -D_REENTRANT -lthread -O -s prvi.c -o prvi`).

Sve opisane funkcije za rad sa dretvama vrijede samo na *Solaris* operacijskim sustavima. Pored njega postoji i POSIX standard za rad sa dretvama, odnosno, funkcije pisane za taj standard. Većina opisanih funkcija postoji i za POSIX, a razlika je većinom u imenu funkcija koje počinju sa *`pthread`*.