Sveučilište u Zagrebu
PMF – Matematički odjel

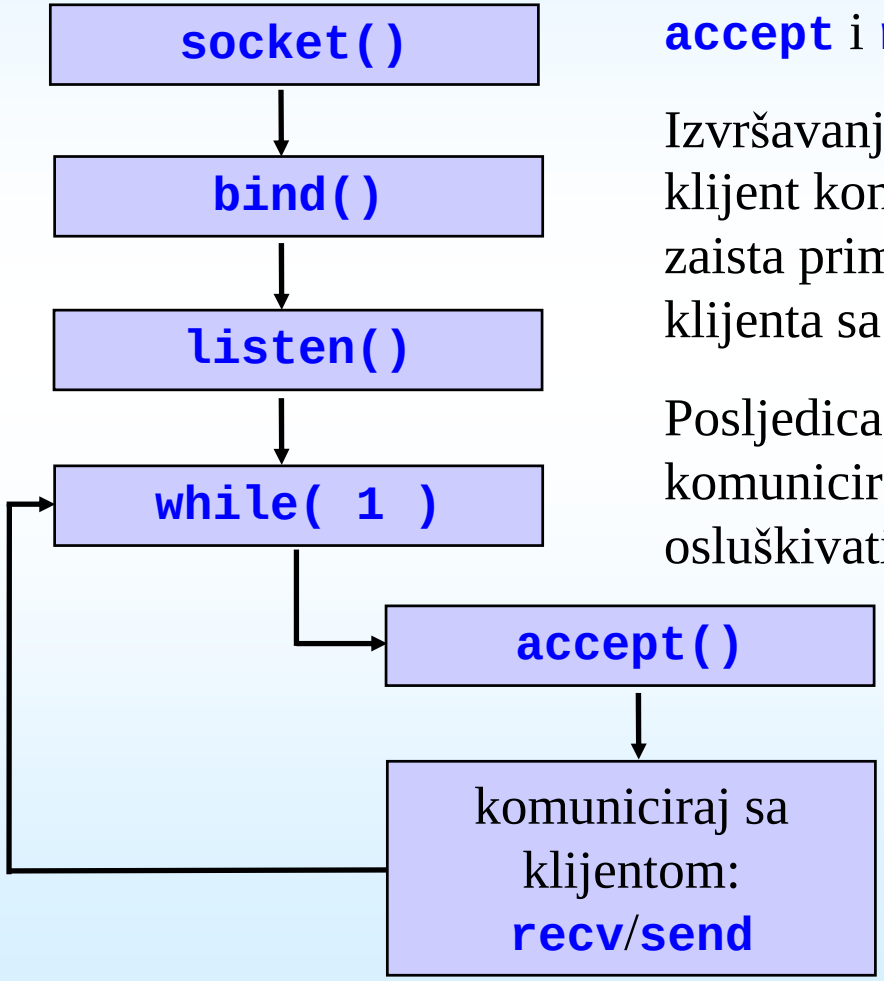
Mreže računala

Vježbe 06

Matko Botinčan
Zvonimir Bujanović
Igor Jelaska
Maja Karaga

Server za više klijenata

- prisjetimo se načina rada servera:



accept i **recv** su "blokirajuće" funkcije.

Izvršavanje programa se nastavlja tek kad neki klijent kontaktira servera (**accept**) ili server zaista primi (**recv**) neku poruku. Isto je kod klijenta sa **recv**.

Posljedica: server može u danom trenutku ili komunicirati samo sa jednim klijentom ili osluškivati dolazak eventualnog novog klijenta.

Više klijenata istovremeno?

- tipičan server treba biti u stanju odjednom komunicirati sa više klijenata. Razlozi:
  - komunikacija sa jednim klijentom može biti dugotrajna (npr. kao kod ftp ili telnet usluge). Ako server odjednom može komunicirati samo sa jednim klijentom, svi ostali moraju dugo čekati na uslugu.
  - server možda služi kao posrednik za interakciju između klijenata (primjer: chat-server ili server za *multiplayer* mrežne igre).

Više klijenata istovremeno?

- načini za ostvarivanje istodobne komunikacije sa više klijenata:
  - naredba `select` – osluškuje više utičnica odjednom, blokira daljnje izvršavanje programa sve dok se na bilo kojoj od njih nešto ne dogodi (bilo pokušaj uspostave komunikacije bilo dolazak podataka od strane nekog klijenta)
  - dizajn servera kao programa koji koristi više *procesa* ili više *dretvi*

- mi ćemo koristiti najfleksibilniji pristup – server će biti *višedretveni* (*multithreaded*) program. Ovaj pristup možemo primijeniti na bilo koji program (ne nužno mrežni).

Višedretveni programi

- moderni operacijski sustavi ostavljaju dojam da odjednom mogu izvršavati više programa:
  - dok pišemo tekst u editoru, možemo istovremeno slušati glazbu, imati pokrenut program koji radi složene matematičke proračune, "skidati" datoteke sa interneta...
  - čak i jedna aplikacija može istovremeno raditi više zadataka, npr. glazbeni *player* istovremeno dekodira glazbu iz mp3 datoteke, pomiče naslov pjesme lijevo-desno po ekranu i iscrtava složenu vizualizaciju u ritmu glazbe

brojanje.c – što je ispis donjeg programa?

```
int j = 0;

void *ispisuj( void *parametar )
{
   int index = *((int *) parametar);
   int i;

   for( i = 1; i <= 20; ++i )
   {
      ++j;
      printf("Funkcija sa parametrom: ");
      printf("%d ispisuje ", index);
      printf("%d; j = %d.\n", i, j);
   }

   return NULL;
}
```

```
int main( void )
{
   int index1, index2;

   index1 = 1;
   ispisuj( &index1 );

   index2 = 2;
   ispisuj( &index2 );

   return 0;
}
```

brojanje.c

- dretva (*thread*) – linija izvršavanja
- program (kao i svi koje smo dosada pisali) koristi jednu *dretvu*
- naredbe se izvršavaju sekvencijalno jedna za drugom:
  - prvo se pozove funkcija `ispisi` sa parametrom `index1`
  - tada se izvršavaju naredbe unutar tijela funkcije; `main` "čeka" sve dok ispisi ne dođe do `return`
  - tada se kontrola ponovno vraća na `main`; on poziva funkciju `ispisi` sa paramerom `index2`
  - `main` ponovno "čeka" sve dok `ispisi` ne dođe do `return` i ne vrati mu kontrolu

pthread

- na UNIX operativnim sustavima za postizanje višenitnosti aplikacija koristi se biblioteka pthread

```
#include <pthread.h>
```

- prilikom kompajliranja programa treba napisati:

```
gcc prog.c –lpthread –o prog
```

Želimo postići da se istovremeno izvršavaju oba poziva funkcije `ispisi`, tj. da istovremeno i jedna i druga funkcija "vrte" for-petlju, mijenjaju globalnu varijablu, te ispisuju na ekran.

pthread_create

- slično kao utičnice kod SocketAPI, biblioteka pthread koristi varijable tipa pthread_t za identificiranje dretve (linije izvršavanja)
- stvaranje nove, *paralelne* dretve:

```
int pthread_create(
   pthread_t *thread, const pthread_attr_t *attr,
   void *(*start_routine)(void *), void *arg );
```

  - thread – identifikator nove dretve (vrijednost će napuniti funkcija)
  - attr – svojstva dretve koju stvaramo, NULL za naše potrebe
  - start_routine – pointer na funkciju koja će biti paralelno pokrenuta
  - arg – pointer na adresu gdje se čuvaju parametri za start_routine
  - vraća 0 ako je uspješno stvorila novu dretvu, inače kodni broj pogreške

brojanje_pthreads.c [main]

```
pthread_t dretva[2];
int bad;

index1 = 1;
bad = pthread_create(
          &dretva[0], NULL,
          ispisuj, (void *)&index1 );
if( bad )
   error( "Greska prilikom kreiranja dretve 1!\n" );

index2 = 2;
bad = pthread_create(
          &dretva[1], NULL,
          ispisuj, (void *)&index2 );
if( bad )
   error( "Greska prilikom kreiranja dretve 2!\n" );
```

brojanje_pthreads.c [main]

- sada naš program ima 3 dretve koje se sve odvijaju paralelno:
  - prilikom pokretanja programa postoji samo 1 dretva (`main`)
  - prva `pthread_create` naredba stvori još jednu dretvu. U toj paralelnoj liniji izvršavanja, izvršavaju se naredbe iz funkcije ispisi. *Istovremeno* sa njima, nastavlja se izvršavanje funkcije `main`.
  - dolaskom na drugu `pthread_create` naredbu, stvara se još jedna dretva, sada ih ima ukupno 3: od tog trenutka nadalje, istovremeno se izvršavaju naredbe iz `main`-a, naredbe iz funkcije `ispisi` sa parametrom `index1 = 1` i naredbe iz funkcije `ispisi` sa parametrom `index2 = 2`.

pthread_join

- kada main dođe do return (što se dogodi prije nego što do return dođu funkcije ispisi), automatski se prekida i izvođenje svih dretvi koje su bile pokrenute iz main
- potrebno je u main-u "pričekati" da ostale dretve završe svoj posao i tek tada izaći. To radi funkcija

```
int pthread_join(pthread_t thread, void **value_ptr);
```

- thread – dretva koju želimo pričekati da završi
- value_ptr – u tu varijablu će biti spremljena povratna vrijednost dretve koju čekamo (ako nas povratna vrijednost ne zanima, pošaljemo NULL)
- povratna vrijednost: 0 za uspjeh, inače kodni broj greške

brojanje_pthreads.c [main]

```
...

   index2 = 2;
   bad = pthread_create(
              &dretva[1], NULL,
              ispisuj, (void *)&index2 );
   if( bad )
      error( "Greska prilikom kreiranja dretve 2!\n" );

   pthread_join( dretva[0], NULL );
   pthread_join( dretva[1], NULL );

   return 0;
}
```

Zadatak 1

- Prepišite programe brojanje.c i brojanje_pthreads.c i usporedite njihove ispise.
- Zbog čega program brojanje_pthreads.c ispisuje "neuredno"?
- Zbog čega se ne javljaju sve vrijednosti između 1 i 40 prilikom ispisa varijable j kod programa brojanje_pthreads.c?

Napomena: ako nema razlike između ispisa programa, iza svake printf naredbe unutar ispisi dodajte npr. (umjesto 1000 možda će trebati i veći broj):

```
while( rand() % 1000 != 0 );
```

Dijeljeni resursi

- dretve programa brojanje_pthreads.c žele istovremeno pristupiti dijeljenim resursima programa:
  - dretve žele istovremeno ispisivati nešto na ekran
  - dretve žele istovremeno koristiti (mijenjati, ispisivati) varijablu j
- da bismo izbjegli konflikte koji zbog toga nastaju, kada neka dretva želi pristup dijeljenom resursu, onda mora osigurati da je ona jedina dretva koja to u tom trenutku radi
- slikovito: na svaki dijeljeni resurs možemo postaviti "lokot"
- kada neka dretva želi koristiti taj resurs, uzet će "ključ" i "zaključati" pripadni "lokot" tako da ga niti jedna druga dretva ne može koristiti. Kada završi sa korištenjem, dretva će "otključati" "lokot" i tako dozvoliti i drugim dretvama da koriste resurs.

Mutex-i

- mehanizam lokota ostvaruje se pomoću mutex-a (*MUTual EXclusion*)
- deklaracija (tipično globalna varijabla):

```
pthread_mutex_t lokot_ekran = PTHREAD_MUTEX_INITIALIZER;
```

- zaključavanje: prije korištenja resursa koristimo funkciju

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- otključavanje: po završetku korištenja resursa koristimo:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- u oba slučaja mutex je lokot na resurs kojeg od- ili za- ključavamo. Funkcije vraćaju 0 u slučaju uspjeha, inače kod greške.

brojanje_pthreads_mutex.c [ispisuj]

```
pthread_mutex_t lokot_ekran = PTHREAD_MUTEX_INITIALIZER;

void *ispisuj( void *parametar )
{
   int index = *((int *) parametar);

   int i;
   for( i = 1; i <= 20; ++i )
   {
      ++j;
      pthread_mutex_lock( &lokot_ekran );
      printf( "Funkcija sa parametrom: " );
      printf( "%d ispisuje ", index );
      printf( "%d; j = %d.\n", i, j );
      pthread_mutex_unlock( &lokot_ekran );
   }

   return NULL;
}
```

Zadatak 2

- Program brojanje_pthreads_mutex.c i dalje pogrešno radi sa globalnom varijablom j.
- Popravite program tako da funkcija ispisi na ispravan način koristi i taj dijeljeni resurs.
- Objasnite razliku koja nastaje kada koristite dva mutex-a i kada koristite samo jedan.

Primjena na serversku aplikaciju

- Sada jednostavno možemo napraviti server koji će raditi istovremeno sa više klijenata:
  - main će biti zadužen isključivo za osluškivanje nadolazećih konekcija, tj. novih klijenata
  - kada dođe novi klijent, sva komunikacija sa njime će se odvijati u zasebnoj funkciji koju ćemo pokrenuti u odvojenoj dretvi.
  - na taj način se istovremeno i osluškuje dolazak novih klijenata i odvija komunikacija sa po volji velikim brojem postojećih

Zadatak 3

- Funkcija `void sleep(int n)` "spava", tj. ne radi ništa n sekundi
- Promijenite daytime_server tako da prije no što pošalje točno vrijeme klijentu "odspava" 20 sekundi.
- Pokušajte se sa 5 klijenata istovremeno (otvorite više terminala) spojiti na modificirani server. Koliko dugo mora zadnji klijent čekati na uslugu?

Zadatak 4

- Promijenite server iz prethodnog zadatka tako da za komunikaciju sa svakim od klijenata koristi zasebnu dretvu.
- Pokušajte se sa 5 klijenata istovremeno (otvorite više terminala) spojiti na modificirani server. Koliko dugo mora zadnji klijent čekati na uslugu?