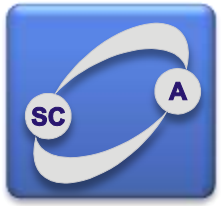


UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA AUTOMATIKU I UPRAVLJANJE SISTEMIMA

Konkurentno i distribuirano programiranje

Distribuirani sistemi

Distribuirano programiranje

Potreba

- Danas razvoj softvera mora uključiti tehnike za implementaciju paralelnog programiranja (PP) i distribuiranog programiranja (DP)
  - Aplikacije treba da se izvršavaju mreži: intranet-u ili Internetu
  - Tipično se traže dobre performanse (brzo izvršavanje)
- Softver mora biti dizajniran da uposli više procesora i/ili više povezanih računara

- Neki primeri zahteva su:
  - Višestruki i jednovremeni *download*-ovi podataka sa Interneta
  - Sofver dizajniran za emitovanje video zapisa mora renderovati grafiku i procesirati zvuk jednovremeno i tečno bez trzavica
  - Web server treba jednovremo da opslužuje veliki broj korisnika
- Da bi ispunila zahteve korisnika savremena softverska rešenja moraju biti složenija i “pametnija” od ranijih.

Izazovi

- Pisanje paralelnih i distribuiranih programa ima 3 izazova:
  - Identifikovanje prirodnog paralelizma u problemu koji se rešava
  - Podela softvera prema zadacima koji se mogu jednovremeno izvršavati
  - Koordinacija tih zadataka da bi izvršavali osnovni zadatak
    - Efikasno!
- Izazove prate poteškoće
  - Deljenje resursa*: Data race*, *deadlock* i njihovo otkrivanje
  - Delimični otkazi i lokalizovanje grešaka
  - Merenje vremena i sinhronizacija zadataka
  - Komunikacione greške i “ćutanje” sagovornika
    Razlike u protokolima
  - Nepostojanje globalnog stanja i centralizovane kontrole resursa
  - ...

Pristup

- Poznavati principe i tehnike programiranja
- Poznavati softverske tehnologije
- Upotrebljavati gotova i proverena rešenja – gde je to moguće
- Dobro razumeti zahteve o ponašanju novog sistema

Konkurentnost

- Za događaje kažemo da su konkurentni kada se pojave u istom vremenskom intervalu (VI)
- Zadaci koji se izvršavaju u istom VI su **konkurentni zadaci**
  - Podrazumeva se da zadaci postoje u istom VI
- Tehnike konkurentnosti se koriste da bi program
  - jednovremeno opsluživao više korisnika
  - uradio više tokom istog perioda
    - Ako se zadaci izvršavaju jedan iza drugog postoje čekanja gde se gubi vreme
  - bio jednostavniji.
- Dobrim dizajnom program se može podeliti na zadatke koji se mogu izvršavati jednovremeno/distribuirano

- Primer: dijeta i vežbanje imaju smisla kada se zajedno sprovode
  - jednovremeno? U istom času: jedemo i vežbamo? U istom periodu – mesecu: prve 2 nedelje jedemo, a naredne vežbamo?

PP, DP i konkurentnost

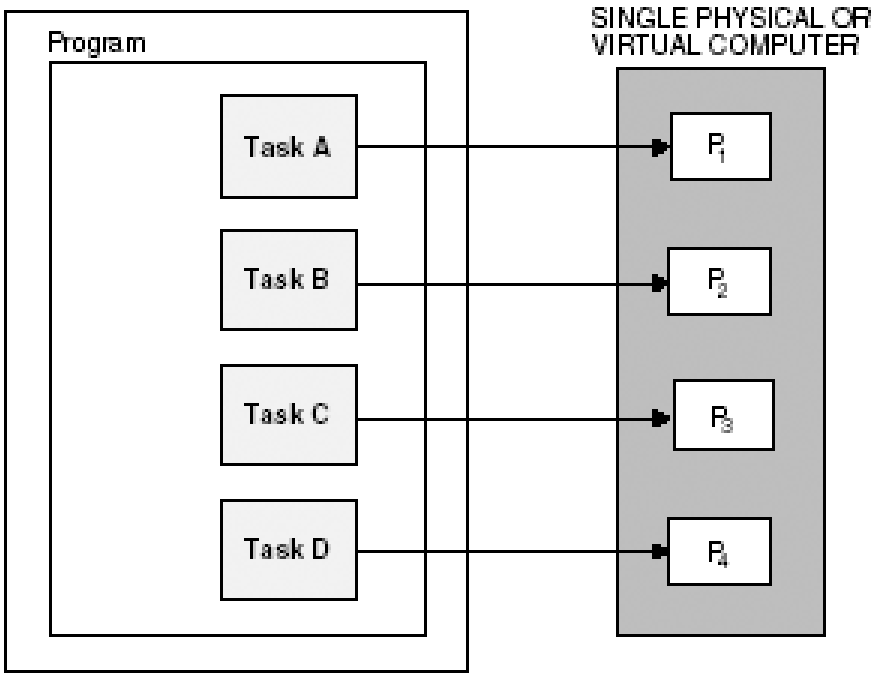
- PP i DP su dva osnovna pristupa za postizanje konkurentnosti
  - Radi se o dve paradigme koje se nekad prepliću

- U PP program koristi 2 ili više procesora unutar istog fizičkog ili virtuelnog računara
- U DP program koristi isti princi PP, ali procesi mogu (a ne moraju) biti u različitim računarima

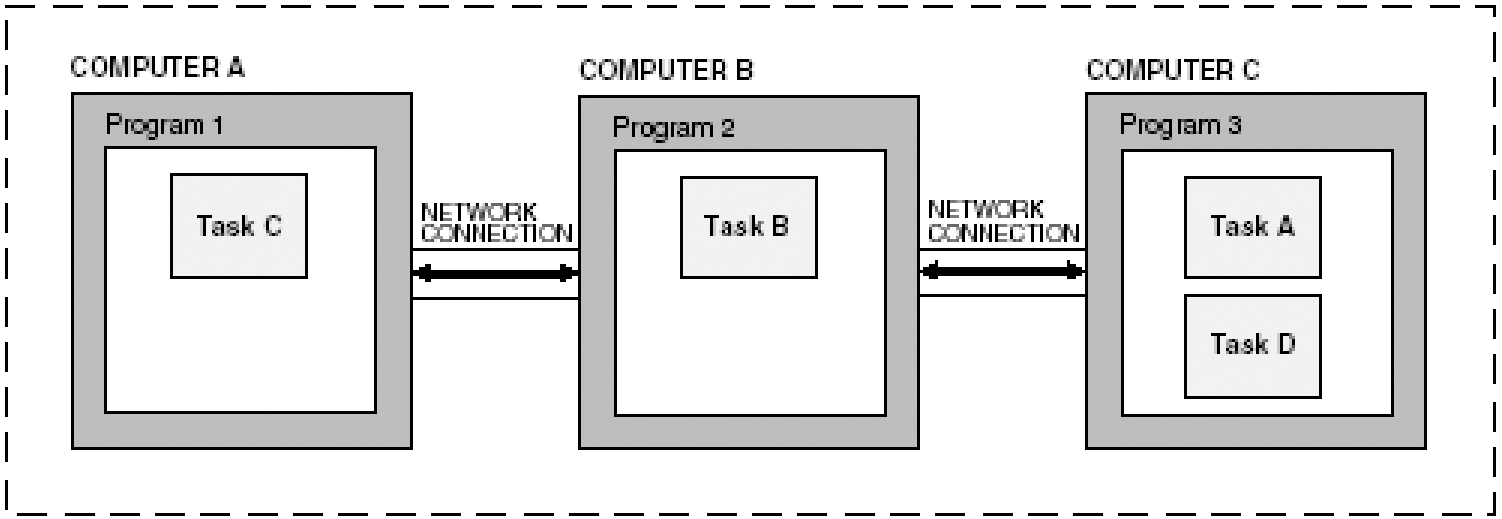
- Paralelni programi su nekad distribuirani
- Distribuirani programi nekad rade u paraleli

- Često programi imaju i paralelno i distribuirano izvršavanja

Paralelni program



Distribuiran program

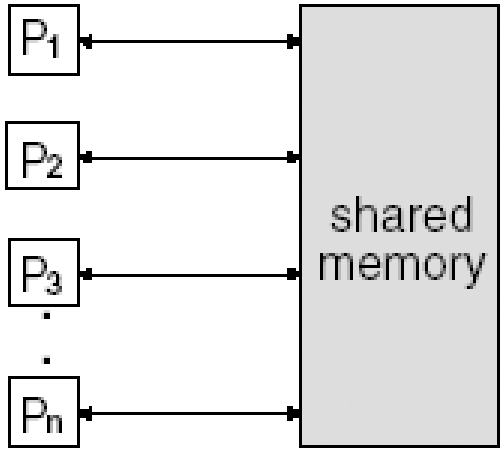


Prednosti PP

- Dobro dizajnirani paralelni program se izvršava brže od njemu odgovarajućeg sekvencijalno organizovanog programa
  - BRŽE == BOLJE
- Neki problemi se prirodno rešavaju kolekcijom paraleleno izvršavanih zadataka
  - Primena u inženjerstvu, nauci, računarskoj inteligenciji, ...
  - Često diktirano specijalizovanim hardverom

PRAM

- Najjednostavniji teorijski paralelni model je PRAM (*Parallel Random Access Machine*)
  - N procesora P1, P2, ... Pn dele zajedničku deljenu memoriju (*shared memory*)
    - Rade nezavisno sem kada pristupaju deljenoj memoriji
  - Osnovne operacije u radu sa deljenom memorijom su **čitanje** i **pisanje** Osnovni *read-write* algoritmi su:
    - ekskluzivno čitanje – ekskluzivno pisanje
    - ekskluzivno čitanje – konkurentno pisanje
    - konkurentno čitanje – ekskluzivno pisanje
    - konkurentno čitanje – konkurentno pisanje



Prednosti DP

- Generalno DP omogućava softveru da se iskoriste resurse locirane u računarskoj mreži

- Deljenje skupih resursa - program na jednom računaru pristupa hardveru drugog računara ili dobija uslugu softvera drugog računara
  - Resursi mogu biti veoma udaljeni (moraju biti povezani u mrežu!)
- Pristup ogromnoj količini informacija koja prevazilazi smeštajne kapacitete jednog računara – lociranoj u raznim čvorovima
- Veća procesorska (računarska) moć je na raspolaganju
- Udvajanje resursa
  - Redundantan hardver (i softver) može rešavati probleme ispada
  - Kopije podataka (replike) omogućavaju brži pristup

Klijent-server model

- Klijent-server model je često upotrebljavan i lako razumljiv distribuiranim sistemima
- Program je podeljen na 2 dela: **server** i **klijent**
  - Server direktno pristupa resursu (hardveru ili softveru) koji treba klijentu
  - Server i klijenti se obično nalaze na raznim računarima
  - Odnos broja servera i klijenata je 1:N, tj. 1 server opslužuje mnogo klijenata
- Server posreduje u pristupu
  - Velikoj količini podataka
  - Skupom ili jedinstvenom hardveru
  - Važnoj kolekciji aplikacija
- Primeri primene su veoma brojni

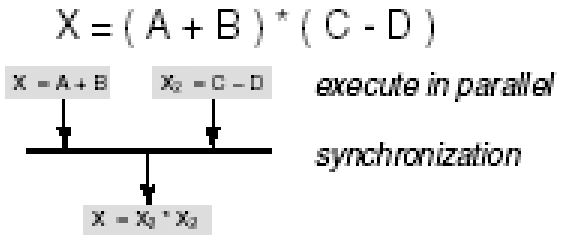
(više u razmatranju arhitekture distribuiranih sistema)

“Cene” paralelizma i distribucije

- PP i DP imaju svoju cenu
- Pre nego se program napiše mora se proći kroz dizajn rešenja koji obuhvata
  - Dekompoziciju – podela programa na delove
  - Komunikacije – neophodne za povezivanje delova
  - Sinhronizaciju – koordinacija rada delova

Nivoi konkurentnosti

- Konkurentnost se može pojaviti na više nivoa:
  - Nivo instrukcija
    - Delovi instrukcije se mogu paralelno izvršiti
  - Nivo funkcije/procedure
    - Funkcionalne celine se izvršavaju jednovremeno (tipično u programskim nitima)



  - Nivo objekta
    - Više objekata učestvuje u rešavanju zadatka. Svaki objekat se izvršava u zasebnom procesu ili niti.
    - Upotrebom CORBA-e objekat se može izvršavati u udaljenom računaru
  - Nivo aplikacije
    - Nekoliko aplikacija može sarađivati u rešavanju istog problema
    - Pojedinačne aplikacije su mogle biti razvijane nezavisno sa drugom namenom, ali su ponovo upotrebljene (*software reuse*) i sinhronizovane za jednu distribuiranu aplikaciju

C++ i paralelizam

- C++ nema podršku za paralelizam
  - Bjarne Stroustrup: “Predlažem da paralelizam ne bude generalna osobina jezika već da se upotrebe biblioteke unutar C++” (slobodan prevod)
- Za paralelizam se koriste sistemske biblioteke (podrška OS) i korisnički definisane biblioteke
- POSIX (*Portable Operating System Interface*) biblioteka niti je skup sistemskih poziva
  - Deo je *Single UNIX Specification* - IEEE Std. 1003.1-2001.
- Neke korisnički definisane biblioteke
  - MPI (*Message Passing Interface*)
  - PVM (*Parallel Virtual Machine*)
  - CORBA (*Common Object Request Broker Architecture*)

Primer

- Izračunati energiju na osnovu izmerenih napona $u(t_k)$ i jačina struje $i(t_k)$ u poznatim sukcesivnim trenucima t_k , k=0,1,...N-1. Merenja se nalaze u tekstualnoj datoteci.
  - Program implementirati u C++ programskom jeziku, a za računanje upotrebiti dve niti

Primer: Organizacija podataka i učitavanje iz datoteke

```
struct SMerenje
{
  float vreme;
  float napon;
  float struja;
};
```

```
void UcitavanjePodataka(char* nazivDatoteke,
                long& brojZapisa, SMerenje*& merenja)
{
  ifstream f(nazivDatoteke);

  float x;
  f >> x;    brojZapisa = long(x);

  merenja = new SMerenje[brojZapisa];

  for ( long k=0; k<brojZapisa; k++ )
    f >> merenja[k].vreme;
  for ( long k=0; k<brojZapisa; k++ )
    f >> merenja[k].napon;
  for ( long k=0; k<brojZapisa; k++ )
    f >> merenja[k].struja;
}
```

Primer: Računanje energije

```
double IzracunajEnergiju ( long brojMerenja, SMerenje* merenja )
{
  double e = 0;
  for ( long k=1; k<brojMerenja; k++ )
    e += (merenja[k].vreme - merenja[k-1].vreme)
          * merenja[k].napon * merenja[k].struja;
  return e;
}
```

```
int main(int argc, char* argv[])
{
  long brojMerenja = 0;
  SMerenje* merenja = NULL;
  UcitavanjePodataka("merenja.dat", brojMerenja, merenja);

  double energija = IzracunajEnergiju(brojMerenja, merenja);
  cout << "Energija = " << energija << endl;
  delete [] merenja;
}
```

Primer: Programska nit

```
class RadnaNit : public omni_thread
{
public:
  RadnaNit(int id, long brojMer, SMerenje* mer, double& rezultat)
            : m_id(id), energija(rezultat), brojMerenja(brojMer), merenja(mer)  {}
  void Start(void) { start_undetached(); }
private:
  void* run_undetached(void*);
  int m_id;
  long brojMerenja;
  SMerenje* merenja;
  double& energija;
};

void* RadnaNit::run_undetached(void*)
{
  energija = IzracunajEnergiju( brojMerenja, merenja );
  exit( (void*)1 );
  return 0;             // ovde nikada ne stize
}
```

Primer: main()

```
void main(int argc, char* argv[])
{
  long brojMer = 0;
  SMerenje* merenja = NULL;
  UcitavanjePodataka("merenja.dat", brojMer, merenja);

  double en1, en2;
  RadnaNit* nit1 = new RadnaNit( 1, brojMer/2, merenja, en1 );
  RadnaNit* nit2 = new RadnaNit( 2, brojMer/2+1, merenja + brojMer/2 - 1, en2 );

  nit1->Start();
  nit2->Start();

  void* p = NULL;
  nit1->join(&p);
  nit2->join(&p);

  cout << "Energija = " << en1 + en2 << endl;
  delete [] merenja;
}
```

Primer: razmišljanja

- Promeniti način računanja energije uz pretpostavku da se i struja i napon linearno menjaju između dva merenja.
- Šta treba promeniti u kodu da rešenje koristi M programskih niti?
- Izmeriti trajanja proračuna za razne brojeve programskih niti.
- Kako bi izgledalo rešenje gde se računanje odvija u realnom vremenu? Dodatna nit se koristi za očitavanje novoizmerenih vrednosti iz uređaja, a tekuće niti treba prepraviti da uvažavaju novopristigla merenja.

Izazovi PP i DP (ponovo)

- Prestati razmišljati isključivo o sekvencijalnom izvršavanju programa
- Koordinacija zadataka
- Hardverski i softverski otkazi
- Preterivanje u paralelizmu
- Izbor pogodne softverske arhitekture
  - Priroda sistema utiče na arhitekturu
- Testiranje rešenja i traženje grešaka
  - Znatno složenije u odnosu na sekvencijalan model programiranja
- Prikladno dokumentovanje rešenja
  - Upotreba UML-a

PP - nova paradigma

- Jednovremenost izvršavanja
  - Više instrukcija se može izvršavati u istom trenutku
  - Jedna instrukcija se može izdeliti na više delova
  - Program se može izdeliti na više zadataka
- Sekvenca događaja i lokacija izvršavanja nisu uvek predvidivi
  - Više zadataka se može pokrenuti jednovremeno ali se ne zna koji će prvi završiti, u kom redosledu će se završitit, na kom procesoru će se izvršavati i sl.
- Sve ovo se znatno usložnjava kada se posmatra DP jer se zadaci mogu izvršavati u raznim računarima
  - Nepostojanje globalnog sata

- Upotreba jednoprocesorskih sistema ne pojednostaljuje razmišljanje jer multitasking savremenih OS vrši virtuelizaciju procesora

Koordinacija zadataka

Zadaci moraju da komuniciraju i sinhronizuju svoj rad

- ***Data race*** – javlja se kada više zadataka menjaju deljeni podatak u isto vreme tako da konačna vrednost zavisi od redosleda pristupa
  - Rešenje: Sinhronizaciju pristupa zasnovati na poznatim pravilima
- **Beskonačno odlaganje** – javlja se kada zadatak čeka na događaj koji se ne desi, tako da on čeka, čeka, ...
- ***Deadlock*** – javlja se kada dva zadatka X i Y žele ekskluzivan pristup do dva resursa A i B, ali zahteve ispostavljaju u suprotnom redosledu. X nakon odobrenja pristupa A čeka na odobrenje za B, ali ga na dobija jer je Y već zauzeo taj resurs (B) i neće ga osloboditi jer čeka da se oslobodi A.
  - Veći broj deljenih resursa i zadataka komplikuje situaciju
- **Komunikacione poteškoće**
  - Komunikacija je nepouzdana; unosi promenljivo kašnjenje
  - Povezivanje heterogenih sistema nosi dodatne komplikacije

Hardverski i softverski otkazi

- U DS se nameće mnoštvo pitanja na koje treba imati spreman odgovor
  - Šta uraditi kada jedan procesor ili računar zakaže?
  - Da li tada program treba da prekine rad ili zadatak posveti drugom?
  - Šta kada komunikacioni link privremeno zakaže? Koliko čekati na odziv kod veoma opterećenog sistema (gustog mrežnog saobraćaja)?
  - Šta uraditi kada deo aplikacije zakaže?
  - Kako promena prava korisnika utiče na celu aplikaciju? Npr. deo aplikacije (koji je ranije radio bez greške) ne može da priđe podacima jer su promenjana prava pristupa.
  - ...

Preterivanje u paralelizmu

- Upošljavanje prevelikog broja procesora (procesa i programskih niti) može imati negativne posledice
  - Komunikacije među procesima u DS se usložnjavaju
  - Prekljućenja na jednom procesoru zahtevaju vreme
  - Složena sinhronizacija procesa
- Optimalan broj potrebnih procesa ili računara je često nepoznat
  - Pomažu: merenja performansi sistema i iskustvo