Višenitno programiranje u Javi

Objektno-orjentisano programiranje

školska 2019/20

AKM

Niti u Javi

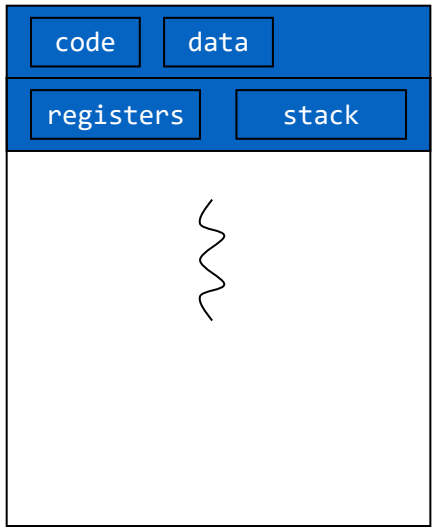
- Niti i višenitno programiranje
- Definisanje, kreiranje i pokratanje niti u Javi – Runnable interfejs, Thread klasa
- Stanja niti

Niti

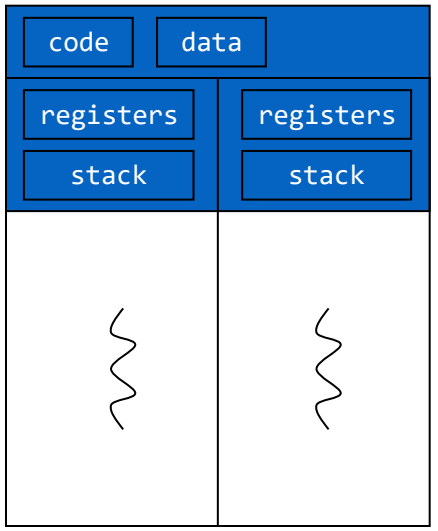
- Nit kontrole (thread) je skup koraka/komandi koji se izvršavaju sekvencijalno/redom
- Nit (thread) predstavlja jedno izvršavanje nekog dela koda unutar adresnog prostora nekog procesa kreiranog unutar OS-a
    - Proces na nivou OS-a (teški proces, heavyweight process): jedno izvršavanje jednog programa sa sopstvenim adresnim prostorom
    - Nit (laki proces, lightweight process): jedno izvršavanje dela koda programa unutar adresnog prostora "okružujućeg" procesa
- Više niti može biti kreirano unutar istog procesa
- Obično se kreiraju nad pozivom jednog potprograma (i svim ugnježdenim pozivima)

Višenitno programiranje

- Niti u okviru istog procesa imaju zajedničko:
  - adresni prostor – globalne podatke
  - programski kod
  - resurse – fajlove i druge sistemske resurse
- Niti u okviru istog procesa imaju sopstveno:
  - poziciju u izvršavanju – PC i trag izvršavanja – stek (SP)
  - podatke lokalne za potprograme – stek (SP) i promenljive alocirane u registre



jednonitni proces



višenitni proces

Višenitno programiranje

- U većini tradicionalnih programskih jezika – samo jedna nit kontrole postoji (C – vezana za main)
- Višenitno programiranje (multi-threading)
  - simultano izvršavanje više niti koje mogu deliti neke zajedničke podatke
- Višenitni program može da se izvršava:
  - konkurentno (ne pretpostavlja više procesora)
  - paralelno (pretpostavlja postojanje više procesora koji omogućavaju stvarni paralelizam)
- Primeri upotrebe - ista aplikacija/proces/program, ali više uporednih aktivnosti-niti:
  - tekst-procesor: snima dokument u pozadini, proverava gramatiku u pozadini, uporedo obrađuje pritiske tastera i druge akcije korisnika
  - Web Browser: dovlači slike ili drugi sadržaj, prikazuje dohvaćeno, obrađuje akcije korisnika
  - server: po jedna nit nad istim programom za svaki zahtev klijenta

Višenitno programiranje

- Koristi:
  - Bolji odziv interaktivne aplikacije: duge aktivnosti se mogu raditi “u pozadini”, a uporedo prihvatati akcije korisnika
  - Deljenje resursa: niti dele memoriju (adresni prostor) i otvorene fajlove
  - Ekonomičnost: kreiranje procesa, promena konteksta, alokacija memorije i ostalih resursa je skupo; niti te režijske troškove ili eliminišu, ili drastično smanjuju; npr. Solaris: kreiranje 30:1, promena konteksta 5:1

Višenitno programiranje

- Podrška nitima može biti izvedena:
  - na nivou korisničkog programa (user threads): niti podržava izvršno okruženje ili biblioteka programskog jezika, OS nema koncept niti;
    - prednosti: efikasnost i jednostavnost
    - mane: kada se jedna nit blokira na sistemskom pozivu, ceo proces se blokira
  - na nivou OS-a (jezgra, kernel threads): OS direktno podržava koncept niti
- Ako su podržane na oba nivoa, modeli preslikavanja:
  - više u jedan (many-to-one): više korisničkih implementira jedna sistemska nit ili proces
  - jedan na jedan (one-to-one): jednu korisničku nit implementira jedna sistemska nit
  - više u više (many-to-many): više korisničkih implementira isti ili manji broj sistemskih niti
    - napomena: sistemska nit/niti implementira korisničku/e

Java niti

- U slučaju Jave i današnjih operativnih sistema niti su podržane i od strane OS-a i od strane Jave
- I kad ne predvidite kreiranje sopstvenih niti, JVM će svakako pokrenuti jednu nit vezanu za main – glavna nit.

Java niti

```
public class TestThread{
   public static void main (String[] args) {
         PingPong nit1 = new PingPong("ping", 33); // 1/30s
         PingPong nit2 = new PingPong(“PONG", 100); // 1/10s
         nit1.start();
         nit2.start();
   }
}
```

upotreba / pokretanje

definisanje niti

```
class PingPong extends Thread {
   public String rec;
   int vreme;PingPong(String r, int v){
         rec=r; vreme=v;
         }
   public void run(){
         try{
                  for(int i=0;i<30;i++){
                           System.out.println(rec);
                           sleep(vreme);}
                  }
                  catch(InterruptedException e){ return; }
         }
}
```

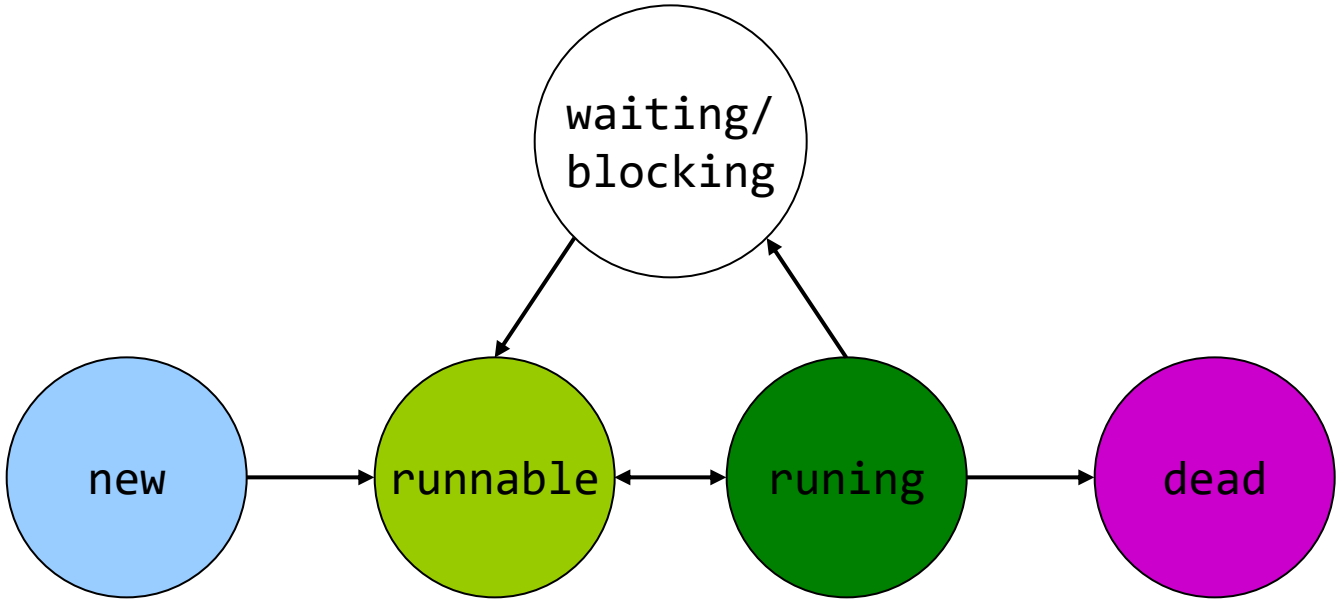
Java niti

- Niti su definisane klasom Thread u standardnoj biblioteci
- Aktivan objekat (sadrži vlastitu nit kontrole) se kreira na sledeći način:

```
Thread nit=new Thread();
```

- Nakon kreiranja niti ona se može:
  - konfigurisati (postavljanjem prioriteta, imena,...)
  - pokrenuti (pozivom startmetode)
- Metod start aktivira novu nit kontrole (nit postaje spremna), a zatim se metod završava i to tako što pokreće njen run metod
- Okruženje (virtuelna mašina) pokreće run metod aktivne niti
  - Standardni metod Thread.run() ne radi ništa
  - Metod run treba da bude redefinisan u korisničkoj klasi koja proširuje klasu Thread
  - Kada se run metod niti završi, nit završava izvršenje

Životni ciklus niti



Stanja niti

```
public class TestThread{
   public static void main (String[] args) {
          PingPong nit1 = new PingPong("ping", 33); // 1/30s
          PingPong nit2 = new PingPong("PONG", 100); // 1/10s
          nit1.start();
          nit2.start();
   }
}
class PingPong extends Thread {
   public String rec;
   int vreme;PingPong(String r, int v){
          rec=r; vreme=v;
          }
   public void run(){
          try{
                   for(int i=0;i<30;i++){
                            System.out.println(rec);
                            sleep(vreme);}
                   }
                   catch(InterruptedException e){ return; }
          }
}
```

nit je kreirana – stanje new

nit je startovana, tj. pokrenut je njen metod run – stanje runnable/running

u zavisnosti od toga da li se trenutno izvršava na procesoru ili čeka na izvršavanje

nit je završila sa radom – stanje dead

Runnable objekti

- Drugi način kreiranja niti – implementacija interfejsa Runnable
  - Interfejs Runnable je apstrakcija koncepta aktivnog objekta - objekta koji izvršava neki kod konkurentno sa drugim takvim objektima
  - Klasa Thread implementira Runnable
  - Problem u kreiranju niti izvođenjem iz klase Thread je to što se sama klasa niti ne može se izvoditi i iz neke druge, ako je tako nešto potrebno onda se projektuje klasa koja implementira interfejs Runnable
  - Ovaj interfejs deklariše samo jedan metod: run
- Postupak kreiranja i korišćenja niti koje su definisane klasama koje implementiraju interfejs Runnable (a nisu izvedene iz klase Thread) :
  - projektovati klasu koja implementira interfejs Runnable
  - kreirati objekat te klase
  - proslediti objekat konstruktoru klase Thread

Runnable objekti

```
public class TestThread{
   public static void main (String[] args) {
         new Thread(new PingPong("ping", 33)).start(); // 1/30s
         new Thread(new PingPong("PONG",100)).start(); // 1/10s
   }
}
class PingPong implements Runnable {
   public String rec;
   int vreme;PingPong(String r, int v){
         rec=r; vreme=v;
         }
   public void run(){
      try{
         for(int i=0;i<30;i++){
                  System.out.print(rec+"\n");
                  Thread.sleep(vreme);}
         } catch(InterruptedException e){
                  return;
                  }
         }
}
```

definisanje

prvo kreirati objekat koji jeste Runnable, tj. koji ima ponašanje definisano u run metodu

a zatim kreirati nit koja će "preuzeti" njegovo ponašanje i osobine

životni ciklus

je isti kao i kod niti koje su definisane kao proširenja klase Thread

`InterruptedException` is thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.