

## Višenitno programiranje u jezicima Java i C# (1)

Osnovna nit koja se provlači kroz rad se može izreći sledećim rečenicama: Opšti programski jezici kao što su ovde navedeni Java i C# imaju istu popularnost i zastupljenost na tržištu i kod programera. Iz toga nameće se stav da su njihove mogućnosti iste. Sintaksa je veoma slična, analogne klase postoje “i tamo i amo”. Poznato je uostalom kako Majkrosoft nastupa na tržištu: prisvajajući najbolje kao svoje. Verovatno Java i C# imaju zato iste višenitne mogućnosti.

*Niti (engl. threads), odnosno laki procesi (engl. lightweight processes), predstavljaju bazične celine za izvršavanje koda pod savremenim operativnim sistemima. Niti su programska celina koja treba da obavi jedan posao. Niti (jedna ili više) pripadaju jednom klasičnom odnosno teškom (engl. heavyweight) procesu. U klasičnom kontekstu jedan težak proces ima svoj programski brojač i druge procesorske registre, memorijske sekcije poput koda, podataka i steka, i ulazno-izlazne resurse.*

Niti kao laki procesi i delovi jednog istog procesa imaju svoje unikatne resurse i zajedničke resurse sa ostalim nitima istog procesa. Od unikatnih resursa imaju poseban identifikator niti (engl. thread ID), posebnu vrednost programskog brojača, vrednosti drugih registara procesora kao i poseban stek. Mnogi moderni softverski paketi su višenitni: programi za prikazivanje Web stranica, programi za obradu teksta itd.

Korišćenje višenitnog koncepta ima sledeće prednosti:

[1] **Manje vremena odziva.** Višenitna tehnika omogućava interaktivnim aplikacijama da nastave rad, čak i kada je deo programa blokiran ili izvršava neku dugotrajnu specifičnu operaciju. Na primer, višenitni Web čitač može da nastavi interakciju s korisnikom u jednoj niti, dok druga nit simultano učitava neku veliku sliku sa Interneta, treća sledeću sliku itd.

[2] **Ekonomičnost.** Ogleda se u deljenju prostora i resursa kao i uštedi vremena koje takođe drastično utiče na performanse. Niti dele memoriju i sve ostale resurse koji pripadaju istom procesu.

[3] **Iskorišćenje višeprocorske arhitekture.** Bilo koje niti mogu se istovremeno izvršavati, svaka na različitom procesoru.

Podrška za korisničke niti realizuje se preko biblioteke za rad s korisničkim nitima. Ova biblioteka obezbeđuje podršku za stvaranje niti, raspored izvršavanja niti i upravljanje nitima, ali bez uticaja jezgra. Najznačajnije vrste korisničkih niti su: POSIX Pthreads, Mac C-threads i Solaris UI-threads.

Niti jezgra direktno podržava operativni sistem. Konkretno, jezgro izvršava operacije stvaranja niti, raspoređivanja izvršavanja niti i upravljanja nitima u prostoru jezgra. Niti jezgra se po pravilu sporije prave, a upravljanje njima unosi veće vremensko premašenje u odnosu na upravljanje korisničkim nitima, ali generalno gledano jezgro ozbiljnije i efikasnije upravlja svojim nitima nego biblioteka za rad s korisničkim nitima.

Mnogi sistemi podržavaju obe vrste niti, a zavisno od toga kako se korisničke niti mapiraju/preslikavaju u niti jezgra postoje tri glavne koncepcije odnosno tri višenitna (engl. multithreading) modela:

[1] Model više u jednu (engl. **many-to-one**). U ovom modelu, više korisničkih niti mapira se u jednu nit jezgra.

[2] Model jedna u jednu (engl. **one-to-one**). U ovom modelu, koji je karakterističan za operativne sisteme Windows NT familije poput 2000, XP i Viste, svaka korisnička nit mapira se u jednu nit jezgra. Ovaj model

obezbeđuje mnogo bolje konkurentno izvršavanje niti, dozvoljavajući da druge niti nastave aktivnosti u slučaju kada jedna nit obavi blokirajući sistemski poziv. Takođe se omogućava da se više niti jezgra izvršavaju paralelno na višeprocorskoj arhitekturi.

**[3] Model više u više (engl. **many-to-many**).** U ovom modelu, više korisničkih niti mapira se u manji ili isti broj niti jezgra pri čemu mapiranje zavisi od operativnog sistema a naročito od broja procesora. Ovo je najkompleksniji i najkvalitetniji model.

Konkurentno programiranje predstavlja pisanje takvih programa koji se sastoje od više kooperativnih procesa i niti koje se izvršavaju simultano ili paralelno, pri tom koristeći zajedničke resurse računarskog sistema. Za pisanje konkurentnih programa mogu poslužiti mnogobrojna razvojna okruženja, među kojima su i programski jezik Java i C#, na koje se odnosi ovaj rad. Smatramo da je reč Java ušla u svakodnevni govor informatičkih stručnjaka i da se ona može pisati na srpskom jeziku, uključujući ćirilično pismo, zbog istog značenja kao u engleskom jeziku. Istovremeno smatramo da to nikako nije slučaj sa nazivom drugog programskog jezika, te ćemo koristiti engleski izvornik – C#.

## Programski jezik Java i niti

Programski jezik Java je osmislio Džejms Gosling (James Gosling) 1991. godine i ona je u vlasništvu firme Sun Microsystems. Predstavlja jezik opšte korisničke i poslovne namene, relativno je jednostavan, ipak dosta moćan ali i platformski nezavisan. Engleski izvornik JAVA predstavlja skraćenicu od Just Another Vague Acronym.

Prednosti Jave koji se navode su sledeći:

- Objektna orijentacija. Podržava sve koncepte objektno orijentisanog programiranja a sintaksa je slična jeziku C++ ali su izbačeni složeni koncepti poput pokazivača.
- Prenosivost. Java programi se prevode u tzv. bajt-kod, koji nije mašinski jezik nijednog konkretnog računara, već se izvršava na JVM. Java Virtuelna Mašina predstavlja virtuelni računar koji može biti simuliran na bilo kom računaru.
- Prirodna prilagođenost Internetu. Java programi mogu da se izvršavaju u Web čitačima. Poseduju sigurnosne mehanizme. Mogu da se distribuiraju i izvršavaju na različitim mašinama.
- Ostale pogodnosti. Podržava konkurentno programiranje. Postoje klase za korisnički interfejs koji omogućuje jedinstven izgled i korišćenje aplikacija.

Ako se za pisanje programa koristi besplatno okruženje Sun JDK (Java Development Kit), onda se Java izvorni kod nalazi u tekstualnim datotekama sa nastavkom .java. Java prevodilac koji izvorni kod kompajlira u bajt-kod se pokreće sa komandom:

```
javac program.java
```

Tako se dobija datoteka sa istim imenom ali nastavkom .class, koju pokreće Java interpreter iz JDK sa komandom:

```
java program
```

## Osnovna nit u Javi

Potrebno je naglasiti razliku između procesa i niti. Svaki proces se nalazi u posebnom memorijskom prostoru, tako da je komunikacija između procesa prilično složena i ograničena. Sa druge strane, niti dele isti adresni prostor u okviru jednog procesa i komunikacija između njih je dosta jednostavnija u odnosu na komunikaciju između

procesa. Radom sa više procesa upravlja operativni sistem, dok radom sa više niti upravlja Java okruženje.

Osnovna ili glavna nit uvek postoji u svakom procesu odnosno aktivnom programu. Kada program u Javi počne da se izvršava, on automatski kreira i izvršava jednu osnovnu nit. Podrška za niti se nalazi u klasi `java.lang.Thread`.

Osnovna nit se kreira na početku `main()` metode. Ova metoda predstavlja uobičajenu tačku ulaska u program. Mora biti deklarirana kao `public static void`, kao i da joj potpis bude upotpunjen sa `String args[]` kao argumentom.

Osnovnu nit instanciramo sa sledećom programskom linijom:

```
Thread osn_nit = Thread.currentThread();
```

Svaka nit ima svoje ime te tako podrazumevano za osnovnu nit glasi `Thread[main,5,main]`. Menjanje se radi sledećom linijom:

```
osn_nit.setName("Osnovna nit");
```

Metoda `getName()` vraća niz znakova koji predstavljaju trenutno ime niti:

```
System.out.println("Ime osnovne niti u Javi glasi: " + osn_nit.getName());
```

Ime niti se može dodeliti i sa navođenjem u konstruktoru. Ostale metode koje se mogu primeniti na osnovnu nit će se opisati malo kasnije.

## Kreiranje niti u Javi

U Javi se nit može napraviti odnosno definisati na dva načina:

1. Implementacijom interfejsa `Runnable` i
2. Proširenjem klase `Thread`.

Prvi način je u opštem slučaju mnogo prikladniji nego izvođenje potklase iz `Thread` klase zato što tako može da se izvede klasa iz neke klase koja nije klasa `Thread`, a da ona i dalje predstavlja nit. Kako Java dozvoljava samo jednu baznu klasu, ako neka klasa se stvori proširenjem `Thread` klase, ona neće imati mogućnost nasleđivanja funkcionalnosti iz neke druge klase.

Jedino što je potrebno je da se implementira metoda `run()` u kojoj se nalazi kod koji će stvarno da izvršava niti.

```
public class Primer_1 implements Runnable {
    public Primer_1() {
        // konstruktor
    }
    public void run() {
        //Metoda koja se izvrsava
    }
    public static void main(String args[]) {
        Primer_1 jedna_nit = new Primer_1();
    }
}
```

```

Thread prva_nit;
prva_nit = new Thread(jedna_nit);
Thread druga_nit = new Thread(new Primer_1());
prva_nit.start(); //startovanje prva niti
druga_nit.start(); //startovanje druge niti
}

```

Ipak run() metodu ne pozivate direktno već prvo se poziva metoda start() koja će da pripremi sve što je potrebno za pokretanje niti.

Ekvivalentan primer kada se proširuje klasa Thread izgleda ovako:

```

public class Primer_2 extends Thread {
    public Primer_2(String str) {
        // konstruktor
        super(str);
    }
    public void run() {
        //Metoda koja se izvrsava
    }
    public static void main(String args[]) {
        Primer_2 prva_nit = new Primer_2("Prva nit");
        Thread druga_nit = new Primer_2("Druga nit");

        prva_nit.start(); //startovanje prve niti
        druga_nit.start(); //startovanje druge niti
    }
}

```

Ovde je iskorišćena mogućnost da se u konstruktoru niti pozove konstruktor natklase super() sa imenom niti kao argumentom. Zbog jednostavnosti u odnosu na prvu mogućnost definisanja niti nadalje će se podrazumevati način preko proširenja klase Thread.

## Stanja niti u Javi

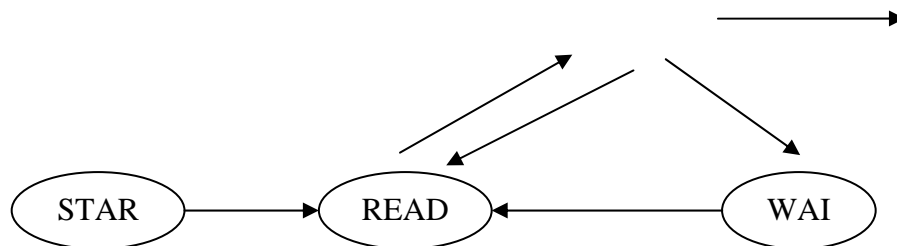
Niti se sastoje iz niza koraka koji slede jedan za drugim. U svakom trenutku nit se nalazi na nekom stanju. Stanje nam govori šta se događa sa niti, šta nit radi ili je u mogućnosti da uradi. Postoji više klasifikacija stanja, a ovde će biti napravljena analogija sa stanjima procesa kod operativnih sistema.

Nit može biti u 5 stanja ako koristimo konačni automat sa istim brojem stanja, a ona su sledeća:

- Start – trenutak formiranja niti, nova nit.
- Ready – nit ima sve potrebne resurse ali čeka na procesor.
- Run – instrukcije niti se izvršavaju.
- Wait – nit čeka neki događaj, miruje, blokirana je, suspendovana je.
- Stop – svršeno stanje, kraj postojanja niti.

RUN

STOP



Dijagram stanja niti

Prevođenje niti iz jednog stanja u drugo se naziva state transition odnosno tranzicija stanja. Broj strelica i njihova usmerenost nam ukazuju na način prevođenja.

U Javi se u stanje start dolazi kad se izvrši naredba:

```
Thread druga_nit = new Primer_2("Druga nit");
```

Prelaz u stanje ready se radi pozivom metode start() za neki nit-objekat:

```
prva_nit.start(); //startovanje prve niti
```

Prelaz u stanje run ne izvodi Java virtuelna mašina već domaćinski (host) operativni sistem, po nekoj od šema za dodelu procesorskog vremena. Kada se nit izvršava (running) onda se u stvari izvršava kod koji se nalazi u run() metodi nit-objekta. Posle isteka izvršavanja rada na procesoru, nit se vraća u stanje ready (runnable, not running).

U stanje wait nit prelazi na više načina:

- Pozivanjem sleep() metode.
- Pozivanjem suspend() metode.
- Pozivanjem wait() metode.
- Pozivanjem blokirajuće U/I operacije.

Metoda sleep() zaustavlja rad tekuće ili imenovane niti zadati broj milisekundi. Za to vreme će se neka druga niti ili više njih izvršavati:

```
prva_nit.sleep(5000); //mirovanje prve niti od 5 sekundi
```

Metoda suspend() blokira odnosno suspenduje izvršavanje tekuće ili imenovane niti, sve dok se ona ne odblokira na jedini mogući način – metodom resume(). Po nekima obe metode su zastarele i više se ne koriste.

```
prva_nit.suspend(); //mirovanje prve niti
// Ovde ide kod koji se izvršava dok je prva nit suspendovana
prva_nit.resume(); //prestanak mirovanja
```

Metoda wait() kao i odblokirajuće verzije notify()/notifyAll() će biti opisane detaljnije u delu o komunikaciji i sinhronizaciji niti.

Pozivanjem operacija koja se odnose na ulazno/izlazne uređaje često dovodi do blokade niti. Takva operacija se neće nastaviti, sve dok se ne oslobodi zauzeti U/I uređaj. Povratak u predhodno stanje se događa kada operacija koja je blokirala U/I uređaj bude završena.

Blokirana nit se može deblokirati jedino odgovarajućom inverznom operacijom od operacije koja ju je blokirala (sleep – done sleeping, suspend – resume, wait – notify, block on I/O – I/O complete). Ako se pokuša deblokirati nit sa neodgovarajućom inverznom operacijom, javiće se `IllegalThreadStateException` izuzetak.

U stanje stop nit dolazi na jedan od četiri načina:

- Metoda je došla do svog prirodnog kraja. Izbegava se tako što se postavlja beskonačna ili uslovna petlja:

```
public void run() {
    while(signal) { //ovde ide kod za izvršavanje }
}
```

- Desio se neuhvaćeni izuzetak. Npr. to se događa sa pozivom `suspend()` metode za nit koja nije u aktivnom stanju, već je u start ili stop stanju. Izbegava se tako što se celo telo `run` metode stavlja u kombinaciju `try/catch` blokova:

```
public void run() {
    try {
        while(signal) { //ovde ide kod za izvršavanje }
    }
    catch(Exception e) { //kod za obradu izuzetka }
}
```

- Kada se nadređena nit ili program u okviru koga je nit kreirana i pokrenuta završi. Izbegava se pravljenjem korisničke niti, koja nastavlja da postoji i posle smrti nadređene niti odnosno nadređenog procesa. Nit koja nije korisnička naziva se pozadinska (daemon) nit. Pozadinska nit se određuje pre poziva metode `start()` sa argumentom `true` za metodu `setDaemon`:

```
Thread druga_nit = new Primer_2("Druga nit");
druga_nit.setDaemon(true); //Odredi da je druga nit tipa pozadinska nit
druga_nit.start(); //startovanje druge niti
```

- Pozivanjem metode `stop()`. Po nekima ova metoda se ne koristi jer dovodi do nestabilnog ponašanja programa:

```
druga_nit.stop(); //kraj postojanja niti
```

Stanje niti se ispituje preko metoda `isAlive()` koji vraća logičku vrednost `true` ako je nit aktivna (bilo da je u stanju `ready`, `run` ili `wait`). Ako nit je stanju `start` ili `stop` onda se vraća vrednost `false`:

```
if(druga_nit.isAlive())
    System.out.println("Druga nit je aktivna!");
else
    System.out.println("Druga nit nije aktivna!");
```

Da li je neka nit korisničkog ili daemon tipa ispituje se sa `isDaemon()` metodom:

```
if(druga_nit.isDaemon())
    System.out.println("Druga nit je daemon tipa!");
else
    System.out.println("Druga nit nije daemon tipa!");
```

Autor: Dragoljub Pilipović