Oblast vežbi : *Konkurentno programiranje*

UVOD U KONKURENTNO PROGRAMIRANJE

Uvod

Višeprocesni ("multitasking") operativni sistem omogućava realizaciju programa sa više programskih procesa koji mogu da se izvršavaju paralelno. U zavisnosti od uzroka koji dovodi do smene tekućeg procesa, sistemi se mogu podeliti na dve onovne grupe:

- **Sistemi bez istiskivanja procesa ("non-preemtive")**
  Kod ove vrste višeprocesnih sistema do smene tekućeg procesa može doći samo ukoliko to zahteva sam tekući proces. Zbog toga je programiranje procesa otežano, pošto procesi moraju povremeno sami pozivati raspoređivača procesa da bi omogućili smenu procesa. Zbog ove osobine ovakvi sistemi se često nazivaju i kooperativni sistemi. Ova vrsta implementacije višeprocesnog okruženja je primenjena kod operativnog sistema Windows 3.x.
- **Sistemi sa istiskivanjem procesa ("preemptive")**
  Ovo su višeprocesni sistemi kod kojih može doći do smene tekućeg procesa i ukoliko to sam proces ne traži. Ovakvi sistemi dodeljuju vremenski interval (*time slice*) za svaki proces. Nakon isteka vremenskog intervala, OS prekida tekući proces, prebacujući ga u stanje pripravan (engl. Ready), i raspoređuje pripravan proces najvišeg prioriteta. Vremenski odsečak zavisi od operativnog sistema i od procesora (približna vrednost na OS Windows 3.x je 20ms). Zbog veoma kratkog vremenskog intervala izgleda kao da se više procesa izvršava u isto vreme. Ovakvi sistemi se često nazivaju i "pravi" multitasking sistemi. Primer njihove primene su operativni sistemi Unix, Linux, Windows 95/98 i Windows NT/2000/XP.

Jedan posao (engl. Job) se sastoji od jednog ili više procesa. Svaki proces može sadržati jednu ili više programskih niti. Niti predstavljaju osnovnu jedinicu operativnog sistema koje konkurišu za vreme procesora. Svaki proces se pokreće izvršavanjem jedne niti koja se naziva glavna programska nit (u programskim jezicima C/C++/Java izvršenje te nit započinje u funkciji “main”). U toku izvršavanja procesa, moguće je stvoriti proizvoljan broj niti iz bilo koje predhodno formirane niti. Svaki proces poseduje resurse potrebne za njegovo izvršavanje (dodeljeni memorijski prostor i druge sistemske resurse). Programske niti dele resurse procesa kome pripadaju. Program se izvršava kada raspoređivač procesa preda upravljanje jednoj od niti procesa koji pripada programu.

Stvaranje programskih niti

Funkcija *CreateThread* stvara novu nit procesa.

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes,
                    DWORD dwStackSize,
                    LPTHREAD_START_ROUTINE lpStartAddress,
                    LPVOID lpParameter,
                    DWORD dwCreationFlags,
                    LPDWORD lpThreadId );
```

Pri stvaranju nove niti potrebno je navesti adresu odakle će početi izvršavanje napravljene niti. To je obično adresa neke funkcije definisane u programu. **Program može imati više niti koje istovremeno izvršavaju kod iste funkcije.**

NAPOMENA: svaka nit zauzima zaseban memorijski prostor za podatke.

Programskoj niti je moguće proslediti pokazivač na bilo koji tip. U slučaju da nema parametara koje treba proslediti, pokazivač se postavlja na NULL.

Funkcija	**Opis**
CreateThread	Stvara novu programsku nit
Parametri	**Opis**
lpThreadAttributes	Sigurnosni atributi (u svim primerima uvek NULL).
dwStackSize	Veličina steka u bajtovima koji se dodeljuje niti (uvek 0, tada se koristi podrazumevana vrednost od 1MB).
lpStartAddress	Adresa funkcije niti. Funkcija se deklariše na sledeći način: `DWORD WINAPI myThread(LPVOID lpvThreadParam);`
lpParameter	Pokazivač na parametar koji će biti prosleđen niti prilikom stvaranja i koji je vidljiv u niti preko **`lpvThreadParam`** parametra (videti opis za **`lpStartAddress`** u vrsti iznad)
dwCreationFlags	Definiše dodatne atribute koji određuju ponašanje niti. Ukoliko je 0 nit ce bit pokrenuta nakon kreiranja. Ukoliko se navede **CREATE_SUSPENDED** atribut, nit **neće** biti pokrenuta nakon njenog formiranja. U tom slu- čaju za pokretanje niti koristi se funkcija ***ResumeThread.***
lpThreadId	Pokazivač na tip DWORD u koji će biti upisan identifikator niti dodeljen od strane operativnog sistema.
Povratna vrednost	**Opis**
Promenljiva tipa Handle	Kreiranje niti je uspelo ako je rukovaoc niti dobio vrednost različitu od NULL.

NAPOMENA:

1. **Redosled kreiranja niti funkcijom *CreateThread* ne mora biti isti sa redosledom pokretanja (kreiranih) niti od strane operativnog sistema.**
2. **I *main* je nit koja se konkurentno izvršava sa ostalim nitima u sistemu. Završetkom funkcije *main*, završava se program.**

Kada se programska nit završi, potrebno je osloboditi kontekst niti funkcijom *CloseHandle.*

```
BOOL CloseHandle(HANDLE hObject);
```

Funkcija	**Opis**
CloseHandle	Oslobađanje kontektsta
Parametri	**Opis**
hObject	Rukovaoc objektom. Objekat može biti nit, semafor, datoteka, dogadjaj, mutex, uređaj, itd.
Povratna vrednost	**Opis**
BOOL	Ako je poziv uspešan, povratna vrednost je različita od nule.

Primer 1 (izvorni kod se nalazi u folderu vezba_a_v1)

Program sa tri programske niti koje ispisuju svoj identifikator na standardni izlaz. Treba primetiti da smenjivanje programskih niti vrši sam operativni sistem. Svaka nit se izvršava određeni vremenski odsečak (engl. *time slice*).

```
#include <stdio.h>
#include <windows.h>

DWORD WINAPI print1(LPVOID lpParam)
{
    for(int i= 0;i<1000;i++)
    {
        printf("1");
    }

    return 0;
}

DWORD WINAPI print2(LPVOID lpParam)
{
    for(int i= 0;i<1000;i++)
    {
        printf("2");
    }

    return 0;
}

DWORD WINAPI print3(LPVOID lpParam)
{
    for(int i= 0;i<1000;i++)
    {
        printf("3");
    }

    return 0;
}

void main(void)
{
    DWORD print1ID, print2ID, print3ID;
    HANDLE hPrint1, hPrint2, hPrint3;
```

```
    hPrint1 = CreateThread(NULL, 0, &print1, NULL, 0, &print1ID);
    hPrint2 = CreateThread(NULL, 0, &print2, NULL, 0, &print2ID);
    hPrint3 = CreateThread(NULL, 0, &print3, NULL, 0, &print3ID);

    int liI = getchar();

    CloseHandle(hPrint1);
    CloseHandle(hPrint2);
    CloseHandle(hPrint3);
}
```

Prethodni primer može se implementirati i sa samo jednom funkcijom (a ne tri kao u prethodnom primeru), kojoj se prosleđuje parameter prilikom kreiranja niti. Ona taj parameter preuzima prilikom startovanja i ispisuje na standardni izlaz. Na osnovu jedne definicije tela niti (jedne funkcije) kreiraju se tri niti, koje imaju zasebne lokalne promenljive, a sve mogu pristupati zajedničkim globalnim promenljivama. Izvorni kod primera se nalazi u folderu vezba_a_v2.

NAPOMENA: Pri pisanju programa sa više niti ako je moguće težiti da se koristi jedna funkcija kao telo niti, a razlike nadomestiti parametrizacijom prosleđivanjem parametra prilikom kreiranja niti)

```
#include <stdio.h>
#include <windows.h>

DWORD WINAPI print(LPVOID lpParam)
{
    char c = *(char *) lpParam;
    for(int i= 0;i<1000;i++)
    {
        printf("%c", c);
    }

    return 0;
}

void main(void)
{
    DWORD print1ID, print2ID, print3ID;
    HANDLE hPrint1, hPrint2, hPrint3;

    char c1 = '1';
    char c2 = '2';
    char c3 = '3';

    hPrint1 = CreateThread(NULL, 0, &print, &c1, 0, &print1ID);
    hPrint2 = CreateThread(NULL, 0, &print, &c2, 0, &print2ID);
    hPrint3 = CreateThread(NULL, 0, &print, &c3, 0, &print3ID);

    int liI = getchar();

    CloseHandle(hPrint1);
    CloseHandle(hPrint2);
```

```
    CloseHandle(hPrint3);
}
```

Sledeći primer (vezba_a_v3) ukazuje na tipičnu grešku prilikom prosleđivanja parametara niti.

```
#include <stdio.h>
#include <windows.h>

DWORD WINAPI print(LPVOID lpParam)
{
    char c = *(char *) lpParam;
    for(int i= 0;i<1000;i++)
    {
        printf("%c", c);
    }

    return 0;
}

void main(void)
{
    DWORD print1ID, print2ID, print3ID;
    HANDLE hPrint1, hPrint2, hPrint3;

    char c;
    // Ovako ne treba raditi
    c = '1';
    hPrint1 = CreateThread(NULL, 0, &print, &c, 0, &print1ID);
    c = '2';
    hPrint2 = CreateThread(NULL, 0, &print, &c, 0, &print2ID);
    c = '3';
    hPrint3 = CreateThread(NULL, 0, &print, &c, 0, &print3ID);

    int liI = getchar();

    CloseHandle(hPrint1);
    CloseHandle(hPrint2);
    CloseHandle(hPrint3);
}
```

Gore navedeni primer ilustruje **pogrešan** način prosleđivanja parametara. Razlog tome je što se koristi samo jedan primerak standardnog tipa (int, char, itd.) ili strukture čija vrednost se prosleđuje niti, a parametar se ne prenosi po vrednosti nego po adresi.

U gore navedenom primeru kreiraju se tri niti, a pre kreiranja date niti, u promenljivu *c* upisuje se odgovarajuća vrednost parametra. Parametar se prenosi preko adrese, što znači da će sve niti dobiti pokazivač na isti objekat, tj. istu adresu kao ulazni parametar.

Formiranje niti se sastoji iz više koraka: rezerviše se memorijski prostor za nit, rezerviše se kontekst niti, i na kraju se pokreće nit. Parametri se najčešće isčitavaju na početku tela niti (funkcije), npr. sa:

char c = *(char *) lpParam;

Do tog momenta, može doći do preključivanja niti, u toku pripreme za formiranje niti. Ukoliko nit *main* opet postane aktivna, ona će promeniti vrednost lokalne promenljive *c* pre no što ju je formirana nit pročitala.

U najgorem slučaju, nit *main* će pokrenuti formiranje svih niti, ijoš uvek će biti aktivna. To dovodi do toga, da kada formirane niti postanu aktivne, *c* će imati vrednost 3, i svaka nit će ispisivati istu vrednost. S obzirom da preključivanje zavisi od momenta startovanja i od operativnog sistema, možda je potrebno pokrenuti primer više puta da bi se postiglo neželjeno ponašanje.

NAPOMENA: Prilikom prosleđivanja parametara nitima, koristiti zasebne primerke struktura ili standardnih tipova, sa svaku formiranu nit ponaosob.

Kritična sekcija

Kritična sekcija obezbeđuje mehanizam nedeljivog pristupa deljenim resursima. Na primer, ako se nekoj strukturi podataka pristupa iz različitih niti sa namerom izmene podataka u strukturi, mora se obezbediti da ne dođe do istovremenog pristupa od strane različitih niti. Drugim rečima operacija izmene podataka prve niti mora se završiti da bi druga nit dobila pravo da menja ili čita te podatke. Ovaj mehanizam nije neophodan jedino u slučaju kada obe niti samo čitaju podatke iz deljene strukture, ali kada postoji i upis, moraju se uvoditi kritične sekcije.

Po zauzimanju kritične sekcije od strane jedne niti ni jedna druga nit je ne može zauzeti dokle god je nit koja ju je zauzela ne oslobodi. Niti pristupaju deljenom resursu po redosledom zauzimanja kritične sekcije.

Funkcije korišćene u radu sa kritičnom sekcijom su:

```
VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Funkcija	**Opis**
InitializeCriticalSection	Inicijalizuje objekat kritične sekcije.
Parametri	**Opis**
lpCriticalSection	Pokazivač na objekat kritične sekcije. Funkcija podrazumeva da je memorija za kritičnu sekciju zauzeta od strane korisnika.
Povratna vrednost	**Opis**
	Nema povratne vrednosti

```
VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Funkcija	**Opis**
EnterCriticalSection	Ulaz u kritičnu sekciju (Zauzimanje kritične sekcije).
Parametri	**Opis**
lpCriticalSection	Pokazivač na objekat kritične sekcije. Funkcija podrazumeva da je objekat kritične sekcije predhodno bio inicijalizovan.
Povratna vrednost	**Opis**
	Nema povratne vrednosti

```
VOID LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Funkcija	**Opis**
LeaveCriticalSection	Izlaz iz kritične sekcije (Oslobađanje kritične sekcije).
Parametri	**Opis**
lpCriticalSection	Pokazivač na objekat kritične sekcije. Funkcija podrazumeva da je objekat kritične sekcije predhodno bio inicijalizovan.
Povratna vrednost	**Opis**
	Nema povratne vrednosti

```
VOID DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Funkcija	**Opis**
DeleteCriticalSection	Oslobađanje resursa kritične sekcije.
Parametri	**Opis**
LPCRITICAL_SECTION *lpCriticalSection*	Pokazivač na objekat kritične sekcije.
Povratna vrednost	**Opis**
	Nema povratne vrednosti

NAPOMENA:

1. Broj zauzimanja kritične sekcije označen sa N, a broj oslobađanja sa M.

Situacija	**Opis**
N = M	**Kritična sekcija pravilno oslobođena**
N > M	**Kritična sekcija i dalje zauzeta**
N < M	**Oslobađanje nezauzete kritične sekcije od strane jedne niti može izazvati beskonačno čekanje druge niti pri pokusaju zauzimanja iste kritične sekcije.**

2. Kada se nit završi, kritična sekcija mora da bude slobodna. U protivnom može doći do zaustavljanja rada ostalih niti ako one čekaju na oslobađanje te kritične sekcije (eng. deadlock).

3. Ne koristiti "return" pre napuštanja kritične sekcije.

4. Biti oprezan prilikom korišćenja funkcije `LeaveCriticalSection` **u okviru nekog uslovnog grananja (if, else) zbog mogućnosti njenog neizvršavanja (u zavisnosti od uslova) i samim tim dovođenja drugih niti, a i samog programa u blokirano stanje (eng. dead lock).**

Primer 2 (vezba_b_v1)

Program sa tri programske niti koje ispisuju tri različite rečenice na standardni izlaz. Postoje dve verzije primera: u prvoj (vezba_1b_v1) se ne koriste kritične sekcije i ispisi rečenica će se preklapati, što je neželjena situacija. Do preklapanja dolazi jer programske niti koriste isti resurs (standardni izlaz) bez međusobne sinhronizacije. Odnosno, ukoliko se izvršavanje progama preključi na sledeću nit, pre nego što je prethodna nit završila sa ispisom svoje rečenice, dolazi do mešanja ispisa.

Da ne bi došlo do preklapanja, uvodi se kritična sekcija (vezba_1b_v2) vezana za standarni izlaz (ispis na ekranu), gde svaka programska nit čeka ulazak u kritičnu sekciju ukoliko je ona zauzeta.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>


CRITICAL_SECTION cs;


DWORD WINAPI print1(LPVOID lpParam)
{
    for(int i=0; i<100; i++)
    {
            EnterCriticalSection(&cs);

            printf("Zvezda");
            printf(" je");
            printf(" prvak");
            printf(" sveta\n");

            LeaveCriticalSection(&cs);
    }

    return 0;
}

DWORD WINAPI print2(LPVOID lpParam)
{
    for(int i=0; i<100; i++)
    {
            EnterCriticalSection(&cs);
```

```
            printf("Priprema");
            printf(" vezbe");
            printf(" iz ");
            printf(" Sistemske");
            printf(" Programske");
            printf(" Podrske\n");

            LeaveCriticalSection(&cs);

    }
    return 0;
}

DWORD WINAPI print3(LPVOID lpParam)
{
    for(int i=0; i<100; i++)
    {
            EnterCriticalSection(&cs);

            printf("Danas");
            printf(" je");
            printf(" lep");
            printf(" i");
            printf(" suncan");
            printf(" dan \n");

            LeaveCriticalSection(&cs);

    }

    return 0;
}

void main(void)
{
    DWORD print1ID, print2ID, print3ID;
    HANDLE hPrint1, hPrint2, hPrint3;


      InitializeCriticalSection(&cs);


    hPrint1 = CreateThread(NULL, 0, &print1, NULL, 0, &print1ID);
    hPrint2 = CreateThread(NULL, 0, &print2, NULL, 0, &print2ID);
    hPrint3 = CreateThread(NULL, 0, &print3, NULL, 0, &print3ID);


      int liI = getch();

    CloseHandle(hPrint1);
    CloseHandle(hPrint2);
    CloseHandle(hPrint3);

      DeleteCriticalSection(&cs);

}
```

Zadatak

Modifikovati drugi primer (vezba_1a_v2) tako da se stvore 3 niti na osnovu jedne funkcije, pri čemu svaka programska nit ima svoj identifikator (1, 2 i 3).

Pomoću kritičnih sekcija i deljenih promenljivih, realizovati mehanizam smenjivanja niti, tako da se na ekranu ispisuje

123123123123123123123123123…….

tj. da se prvo aktivira nit 1, pa nit 2 a zatim nit 3, 100 puta. Posle 100 ispisa, niti se završavaju. Mehanizam za smenjivanje niti treba da obezbedi da nit, nakon svog ispisa signalizira narednoj niti da može da ispiše svoj identifikator.

Zadatak realizovati pomoću kritične sekcije i deljene promenljive. Sve tri programske niti realizovati istom funkcijom kojoj se prosleđuje identifikator programske niti.

Ukoliko se između dva pristupa kritičnoj sekciji umetne pauza (pomoću funkcije Sleep), program će se brže završiti. Zašto?

Dodatak

Kratko upustvo za formiranje projekta

1. Pokrenuti *Microsoft Visual C++*. Ukoliko nemate na Desktop-u odgovarajuću ikonicu program pokrenite sa lokacije
   *\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\MSDEV.EX*E.
2. Izborom stavke **New** iz menija **File** otvorite prozor **Ne**w. Otvorite tabelu **Projects** a zatim u njoj izaberite stavku ***Win32Console Application***. U polju **Project name** pišite ime projekta: npr. *Hello*. U polju **Location** upišite ili odaberite ime direktorijuma u kome će vaš novi projekat biti formiran. Zatim pritisnite **O**K. U prozoru **Win32 Console Application – Step 1 of1** koji se zatim pojavljuje odaberite opciju **An empty project** i pritisnite **Finish**.
3. Sledi dodavanje nove datoteke u projekat. Odaberite ponovo stavku **New** iz menija **Fil**e. Otvorite tabelu **Files** u prozoru **New** i u njoj izaberite tip datoteke koju želite da dodate u projekat (najčešće ***C++ source file*** ili ***C/C++ Header fil***e). Uključite stavku **Add to project** a u "combo" meniju koji se nalazi ispod ove mogućnosti odaberite ime vašeg projekta *(Hello*). U polju **File name** upišite ime nove datoteke sa proširenjem (npr. *Hello*.c). U polju **Location** upišite ili odaberite ime direktorijuma u kome će vaša nova datoteka biti formirana. Napomena: Pravilan izbor proširenja u imenu datoteke je veoma bitan! Npr. ukoliko dodelite proširenje *cpp* datoteci koja sadrži korektan C programski kod, prevodilac će smatrati da se u njoj nalazi C++ programski kod, što može dovesti do pojave sintaksnih grešaka u fazi prevođenja.
4. Unesite programski kod u datoteku *Hello*.c.
   Npr:
   ```
   #include <stdio.h>
   void main( void )
   {
   printf( "Hello World!\n" );
   }
   ```
   Zatim snimite datoteku izborom stavke **Save** iz menija **Fil**e.
5. Parametre projekta možete podešavati u prozoru **Project settings** koji otvarate izborom stavke **Settings...** iz menija **Projec**t. Npr. da bi ste dodali novu biblioteku u projekat (recimo winmm.lib) otvorite tabelu **Link** u ovom prozoru, zatim iz "combo" menija **Category** odaberite stavku **General** i u polje **Object library/modules** dopišite winmm.lib.
6. Prevođenje i uvezivanje programa pokrećete izborom stavke **Build <ime projekta.exe>** iz menija **Build** (u ovom slučaju **Build** / **Build Hello.exe**). Rezultat ove operacije biće vam prikazan u donjem delu ekrana u **Output** prozoru. Ukoliko ste program iz primera uneli bez greške ovaj prozor će imati sledeći sadržaj:

   ```
   --------------------Configuration: Hello - Win32 Debug--------
   Compiling...
   Hello.c
   Linking...
   Hello.exe - 0 error(s), 0 warning(s)
   ```
   U protivnom, program će sadržati obaveštenja o sintaksnim greškama ili upozorenjima. Ukoliko kliknete mišem na liniju sa opisom greške editor će označiti liniju programskog koda u kojoj se greška nalazi.
7. Izborom stavke **Execute <ime projekta.exe>** iz menija **Build** (u ovom slučaju **Build** / **Execute Hello.exe**) pokrećete prethodno preveden i uvezan program.

Program iz primera će otvoriti novi prozor za izvršavanje konzolne aplikacije i u njemu ispisati:

```
Hello World!
Press any key to continue
```

8. Ukoliko želite da pokrenete debuger pritisnite taster **F**5. Prethodno postavite prekidnu tačku na mestu gde želite da se vaš program zaustavi. Jedan od načina da se postavi prekidna tačka je da se pozicionirate kursorom na liniju koda u kojoj želite da postavite prekidnu tacku, a zatim pritisnete taster **F**9. Nakon pokretanja debuger-a komande za rad sa debuger-om dostupne su iz menija **Debug**. U prozorima **Variables** i **Watch** koji se nalaze u dnu ekrana možete pratiti sadržaj programskih promenljivih.
9. Pre izlaska iz programa *Microsoft Visual C++* snimite vaš projekat izborom stavke **Save Workspace** iz menija **Fil**e.