Oblast vežbi : *Konkurentno programiranje*

SINHRONIZACIJA PROGRAMSKIH NITI

Semafori

Semafori su još jedan od mehanizama konkurentnog programiranja. Koriste se za sinhronizaciju izvršavanja niti u okviru jednog procesa.

S je nenegativan ceo broj, nad kojim su definisane dve operacije – UVECAJ(S) i UMANJI(S). Kada je je broj S veći od nule (S > 0) semafor je signaliziran. Kada S ima vrednost nula (S = 0) semafor nije signaliziran. Vrednost S ne može biti manja od nule.

Operacija UVECAJ(S): promenljiva S se uvećava za jedan jednom nedeljivom operacijom. Nedeljivost se obezbeđuje na sistemskom nivou.

Operacija UMANJI(S): umanjenje vrednosti S za jedan jednom nedeljivom operacijom. Umanjenje se vrši ukoliko je to moguće (S > 0). Ako je S=0, umanjenje nije moguće. U praksi semafor se realizuje kao C struktura koja definiše S i red čekanja na semaforu.

Funkcije korišćene u radu sa semaforom su:

```
HANDLE CreateSemaphore(LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
                       LONG lInitialCount,
                       LONG lMaximumCount,
                       LPCTSTR lpName);
```

Funkcija	**Opis**
CreateSemaphore	Stvara novi semafor.
Parametri	**Opis**
lpSemaphoreAttributes	Sigurnosni atributi (u našim primerima uvek NULL).
lInitialCount	Početna vrednost brojača.
lMaximumCount	Maksimalna vrednost brojača.
lpName	Naziv semafora.
Povratna vrednost	**Opis**
Promenljiva tipa Handle	Kreiranje semafora je uspelo ako je rukovaoc semafora dobio vrednost različitu od NULL.

```
DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);
```

Funkcija	**Opis**
WaitForSingleObject	Čekanje na ostvarenje nekog događaja. U slučaju semafora čekanje na pozitivnu vrednost brojača semafora. Ako semafor nije signaliziran (ima vrednost 0), nit se blokira ili do isteka vremena čekanja ili dok vrednost semafora ne postane pozitivna. Ukoliko se kao prvi parametar prosleđuje kontekst niti, izvršavanje tekuće niti se blokira do završetka niti čiji je kontekst prosleđen. Ovaj mehanizam se efikasno može koristiti za regularan završetak projekta.
Parametri	**Opis**
hHandle	Rukovaoc (*Handle*) semafora. (Vraća funkcija *CreateSemaphor*e).
dwMilliseconds	Vreme koje se čeka na semaforu u mili sekundama. Ukoliko se čeka neograničeno prosleđuje se konstanta INFINITE.
Povratna vrednost	**Opis**
Promenjiva tipa DWORD	Ova funkcija može vratiti jednu od 4 povratne vrednosti: **WAIT_ABANDONED** – pojavljuje se u slučaju MUTEX-a koji neće biti dalje razmatran **WAIT_OBJECT_0** – očekivani događaj se ostvario **WAIT_TIMEOUT** – vreme čekanja je isteklo, a događaj se nije ostvario **WAIT_FAILED** – poziv funkcije je neuspeo

```
BOOL ReleaseSemaphore(HANDLE hSemaphore,
                      LONG lReleaseCount,
                      LPLONG lpPreviousCount);
```

Funkcija	**Opis**
ReleaseSemaphore	Oslobađanje semafora.
Parametri	**Opis**
HANDLE *hHandle*	*Handle* semafora. (Vraća funkcija *CreateSemaphor*e).
LONG *lReleaseCount*	Broj za koji se vraća vrednost brojača semafora.
LPLONG *lpPreviousCount*	Prethodna vrednost brojača semafora.
Povratna vrednost	**Opis**
Promenljiva tipa BOOL	Ako poziv uspe povratna vrednost je različita od nule.

```
DWORD WaitForMultipleObjects(DWORD nCount,
                             CONST HANDLE *lpHandles,
                             BOOL fWaitAll,
                             DWORD dwMilliseconds);
```

Funkcija	**Opis**
WaitForMultipleObjects	Čekanje na ostvarenje nekog od događaja. U slučaju semafora čekanje na pozitivnu vrednost nekog semafora ili svih semafora u nizu na koji pokazuje *lpHandles.*
Parametri	**Opis**
DWORD *nCount*	Broj objekata u nizu na koji pokazuje ***lpHandles***
HANDLE *lpHandles*	Pokazivač na niz signalnih objekata (najčešće semafori).
BOOL *fWaitAll*	Specificira tip čekanja: - **TRUE** – čeka se na pozitivnu vrednost svih objekata u nizu - **FALSE** – čeka se na pozitivnu vrednost bilo kog objekta u nizu
DWORD *dwMilliseconds*	Vreme koje se čeka na semaforu u mili sekundama. Ukoliko se čeka neograničeno prosleđuje se konstanta **INFINITE**.
Povratna vrednost	**Opis**
Promenjiva tipa DWORD	Ova funkcija može vratiti jednu od 3 povratne vrednosti: **WAIT_OBJECT_0 do (WAIT_OBJECT_0 + nCount – 1) –** ukoliko je *bWaitAll* **TRUE** povratna vrednost pokazuje da su svi objekti sinhronizacije signalizirani. Ukoliko je *fWaitAll* **FALSE** povratna vrednost funkcije smanjena za **WAIT_OBJECT_0** ukazuje na indeks objekta (semafora) u nizu koji signaliziran. **WAIT_ABANDONED_0 do (WAIT_ABANDONED_0 + nCount – 1) –** pojavljuje se u slučaju MUTEX-a koji neće biti dalje razmatran. **WAIT_TIMEOUT –** vreme čekanja je isteklo, a događaj se nije ostvario.

Primer 1 (vežba_2a)

Programska nit koja čeka pritisak tastera na tastaturi i za svaki pritisak tastera povećava brojač (counter) za jedan. Pritiskom na taster 'q' program završava sa radom.

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


HANDLE hSemaphore;


int counter;

/* Telo niti. */
DWORD WINAPI ThreadProc(LPVOID lpParam)
{
      while(true)
      {
            /* Cekaj na signal da je pritisnut taster. */
            WaitForSingleObject(hSemaphore,INFINITE);
            /* Povecaj brojac i ispisi vrednost. */
            counter++;
            printf("\r counter = %d",counter);
      }
}


void main(void)
{
    DWORD threadID;
    HANDLE hThread;

    /* Inicijalizacija sistema. */
      hSemaphore = CreateSemaphore(0,0,1,NULL);
      hThread  = CreateThread(NULL, 0, &ThreadProc, NULL, 0,
                              &threadID);

    /* Glavna petlja programa, zavrsava se pritiskom na q. */
      while(true)
    {
            /* Da li je pritisnut taster? */
            if(kbhit()){
                  /* Ukoliko jeste, preuzmi ga. Ako je preuzeti znak q,
                  zavrsi program. */
                  if(getch() == 'q') break;
                  /* Obavesti nit da je pritisnu taster. */
                  ReleaseSemaphore(hSemaphore,1,NULL);
            }
            /* Pauza, da procesor ne bi konstantno bio zauzet
              ispitivanjem da li je pritisnut taster. */
            Sleep(100);
    }
```

```
      /* Oslobadjanje zauzetig resursa. */
      CloseHandle(hThread);
      CloseHandle(hSemaphore);
}
```

NAPOMENA:

1. Prilikom kreiranja obratiti pažnju na inicijalnu vrednosti semafora. Ako je inicijalna vrednost veća od nule imati u vidu da je taj semafor već signaliziran i da će nitima koje čekaju na njegov signal biti omogućen dalji rad.

2. Uvaćavanje vrednosti semafora (funkcija `ReleaseSemaphore`) nije moguće preko vrednosti zadate kao maksimalna vrednost brojača prilikom kreiranja semafora. Svi pozivi `ReleaseSemaphore` koji bi doveli do premašivanja maksimalne vrednosti brojača se ignorišu.

Primer 2 (vežba_2aa)

U primeru 2 prikazano je rešenje Vežbe 1, primenom semafora. U glavnoj funkciji *main* formiraju se tri niti koje izvršavaju istu funkciju *print*. Svaka od niti čeka signalizaciju svog semafora. Pri formiranju semafora, inicijalna vrednost prvog semafora u nizu je jedan, da bi niti uopšte otpočeli sa izvršavanjem.

Za razliku od rešenja sa kritičnim sekcijama, gde svaka nit pristupa deljenoj promenljivoj, proverava njenu vrednost i u zavisnosti od nje ispisuje svoj ID, sinhronizacija u ovom primeru je obezbeđena semaforima. Nit nema potrebe da stalno ispituje uslov i na taj način da okupira procesorsko vreme, već će ga sistem obavestiti kada može da otpočne sa izvršavanjem.

NAPOMENA:

Semafori obezbeđuju mehanizam signalizacije između niti a kritične sekcije nedeljiv pristup deljenom resursu.

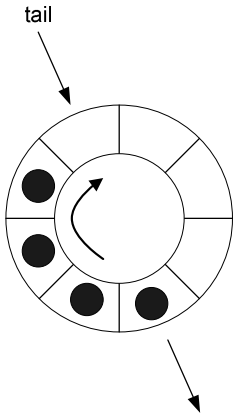
```
#include <stdio.h>
#include <windows.h>
#include <conio.h>


HANDLE hSemaphores[3];

/* Makro za bezbedno brisanje handle-ova.*/
#define SAFE_DELETE_HANDLE(a)  if(a){CloseHandle(a);}

DWORD WINAPI print(LPVOID lpParam)
{
    int n = (int) lpParam;
      for (int i=0; i<1000; i++) {
        /* Svaka nit ceka signalizaciju na svom semaforu i
```

```
           signalizira semafor sledecoj niti.*/
            WaitForSingleObject(hSemaphores[n], INFINITE);
            printf("%d", n + 1);
            ReleaseSemaphore(hSemaphores[(n+1)%3],1, NULL);
      }
      return 0;
}


void main(void)
{
      DWORD print1ID, print2ID, print3ID;
    /* Indetifikatori niti.*/
      HANDLE hPrint1, hPrint2, hPrint3;

    /* Formiranje semafora za niti;
       Svaka nit ima svoj semafor;
       Semafor prve niti je inicijalno postavljen
       na jedinicu da bi niti otpocele sa radom.*/
      hSemaphores[0] = CreateSemaphore(0, 1, 1, NULL);
      hSemaphores[1] = CreateSemaphore(0, 0, 1, NULL);
      hSemaphores[2] = CreateSemaphore(0, 0, 1, NULL);

    /*  */
    if(hSemaphores[0] && hSemaphores[1] && hSemaphores[2]){
        /* Formiranje niti;
           Sve niti imaju isto telo funkcije.*/
          hPrint1=CreateThread(NULL, 0, &print, (LPVOID)0, 0,
                               &print1ID);
          hPrint2=CreateThread(NULL, 0, &print, (LPVOID)1, 0,
                               &print2ID);
          hPrint3=CreateThread(NULL, 0, &print, (LPVOID)2, 0,
                               &print3ID);

        /* Blokirajuca funkcija;.*/
          int c=getch();
    }

    /* Zatvaranje handle-ova.*/
      SAFE_DELETE_HANDLE(hSemaphores[0]);
      SAFE_DELETE_HANDLE(hSemaphores[1]);
      SAFE_DELETE_HANDLE(hSemaphores[2]);
      SAFE_DELETE_HANDLE(hPrint1);
      SAFE_DELETE_HANDLE(hPrint2);
      SAFE_DELETE_HANDLE(hPrint3);
}
```

Primer 3 (vežba_2b)
Proizvođač – Potrošač

Klasičan problem sinhronizacije dve programske niti opisan je problemom proizvođač - potrošač .

Pretpostavimo da imamo kružni bafer (engl. circular buffer) sa dva pokazivača tail i head, gde pokazivač tail pokazuje na prvu slobodnu poziciju u baferu u koju se može smestiti podatak, a pokazivač head ukazuju na poziciju sa koje se čita uskladšteni podatak Neka su dve programske niti (proizvođač i potrošač) vezane preko iste memorijske lokacije u ovom slučaju kružnog bafera. Prva programska nit proizvođač (engl. producer) prozvodi podatke i unosi ih u bafer, dok ih potrošač (engl. consumer) čita iz bafera i koristi. Pošto dve različite programske niti pristupaju istom memorijskom resursu (kružni bafer) potrebno je sinhronizovati pristup tom deljenom resursu. Sinhronizacija se uspostavlja uvođenjem kritične sekcije za operacije vezane za rad sa deljenim resursom. Takođe programska nit proizvođač treba da se blokira ukoliko je bafer pun odnosno programska nit potrošač treba da se blokira ukoliko je bafer prazan.



Kružni bafer
- *tail* pozicija na koju se upisuje podatak
- *head* pozicija sa koje se čita

PROIZVOĐAČ

- proizvodi podatak
- čeka ukoliko je bafer pun
- ulazi u kritičnu sekcju
- unosi podatak u bafer na poziciju *tail*
- izlazi iz kritične sekcije

POTROŠAČ

- čeka ukoliko je bafer prazan
- ulazi u kritičnu sekciju
- čita podatak iz bafera sa pozicije *head*
- izlazi iz kritične sekcije

Realizacija

Osnovna programska nit *main*

U osnovnoj programskoj niti main formiraju se dva semafora Empty i Full. Početna vrednost Empty semafora se je *RING_SIZE* i ukazuje na broj slobodnih mesta u kružnom baferu. Semafor Full se formira sa početnom vrednošću nula i ukazuje na broj popunjenih mesta u kružnom baferu. U main programskoj niti pokreću se dve programske niti: potrošač i proizvođač. Programska nit main nakon pokretanja programskih niti i provera da li su niti formirane, odlazi u stanje čekanja dok se programske niti ne završe. Naime, ukoliko se funkcija WaitForSingleObject poziva za programsku nit (parametar funkcije je objekat handle, odnosno logički broj niti), nit koja poziva ovu funkciju (u posmatranom primeru main programska nit) prelazi u stanje čekanja sve dok se rad programske niti ne završi.

```
/* Glavna programska nit koja formira  dve programske (proizvodjac i
potrosac) niti i ceka njihovo gasenje.*/
void main() {
    /* Identifikatori niti.*/
    HANDLE hProducer;
    HANDLE hConsumer;
    DWORD ProducerID;
    DWORD ConsumerID;

    /* Formiranje Empty Full i ThreadFinishSignal semafora.*/
    Empty           = CreateSemaphore(0,RING_SIZE,RING_SIZE,NULL);
    Full            = CreateSemaphore(0,0,RING_SIZE,NULL);
    FinishSignal    = CreateSemaphore(0,0, 2,NULL);

    /* Provera da li su semafori formirani.*/
    if (Empty && Full && FinishSignal){
        /* Inicijalizacija kriticne sekcije B.*/
        InitializeCriticalSection(&BufferAccess);

        /* Formiranje programskih niti: proizodjac i potrosac.*/
        hProducer =
          CreateThread(NULL,0,&producer,(LPVOID)0,0,&ProducerID);
        hConsumer =
          CreateThread(NULL,0,&consumer,(LPVOID)0,0,&ConsumerID);

      /* Provera da li su niti formirane;Ukoliko funkcija CreateThread
         vrati 0, nit nije formiana.*/
        if(!hProducer || !hConsumer){
          /* Zaustavljanje niti;Vrednost semafora se inkrementira za 2
            da bi signaliziralo obema nitima.*/
            ReleaseSemaphore(FinishSignal,2,NULL);
        }
        /* Cekanje na zavrsetak formiranih niti.*/
        if(hConsumer)
            WaitForSingleObject(hConsumer,INFINITE);

        if(hProducer)
            WaitForSingleObject(hProducer,INFINITE);
    }
```

```
    // Oslobadjanje logickih brojava koji su bili dodeljeni nitima i
       semaforima
    SAFE_DELETE_HANDLE(hProducer);
    SAFE_DELETE_HANDLE(hConsumer);
    SAFE_DELETE_HANDLE(Empty);
    SAFE_DELETE_HANDLE(Full);
    SAFE_DELETE_HANDLE(FinishSignal);

    // Brisanje semafora
    DeleteCriticalSection(&BufferAccess);
}
```

Kružni bafer

Kružni bafer je predstavljen strukturom RingBuffer.
Sastoji se od indeksa:

- tail - predstavlja poziciju na koju se smešta novi podatak u niz data
- head pozicija sa koje se čita podatak iz niza data
- data niz ASCII znakova, veličine *RING_SIZE*

Operacije za rad sa kružnim baferom su definisane kao funkcije:

- ringBufGetChar(r) operacija za čitanje podatka iz niza sa pozicije head
- ringBufPutChar(r, c) operacija za smeštanje podatka u niz na poziciju tail

Prilikom pristupa podacima (ASCII znaci) u nizu data, indeksi tail i head se dele po modulu sa veličinom niza *RING_SIZE* kako bi indeksi bili u granicama veličine niza.

```
//  Kruzni bafer - FIFO
struct RingBuffer {
    unsigned int  tail;
    unsigned int  head;
    unsigned char data[RING_SIZE];
};

//  Operacije za rad sa kruznim baferom
char ringBufGetChar(RingBuffer *apBuffer) {
    int index;
    index = apBuffer->head;
    apBuffer->head = (apBuffer->head + 1) % RING_SIZE;
    return apBuffer->data[index];
}

void ringBufPutChar(RingBuffer *apBuffer, const char c) {
    apBuffer->data[apBuffer->tail] = c;
    apBuffer->tail = (apBuffer->tail + 1) % RING_SIZE;
}
```

Proizvođač

Programska nit proizođač čeka ili da potrošač preuzme podatak iz bafera i poveća *Empty* semafor za jedan ukoliko je semafor *Empty*=0 (kružni bafer pun) ili signalizaciju za završetak rada niti (*FinishSignal*). Početna vredost semafora *Empty* je jednaka veličini bafera *RING_SIZE*. Ukoliko kružni bafer nije pun (*Empty* različit od nule), proizvođač preuzima karakter sa tastature i pokušava da uđe u kritičnu sekciju *BufferAccess* funkcijom *EnterCriticalSection*. Ukoliko je neka druga programska nit, u ovom slučaju potrošač, zauzela kritičnu sekciju prelazi u stanje čekanja sve dok programska nit potrošač ne oslobodi kritičnu sekciju funkcijom *LeaveCriticalSection*. Kritična sekcija sinhronizuje rad programskih niti sa deljenim resursom (kružnim baferom). Nakon unošenja ASCII znaka u bafer potrošač povećava semafor *Full* za jedan.

Ukoliko je pritisnuti znak ‘q’ ili 'Q' programska nit proizvođač signalizira obema nitima da završe sa radom, povećavajući vrednost semafora *FinishSignal* za dva.

```
/* Funkcija programske niti proizvodjaca.*/
DWORD WINAPI producer(LPVOID param){
    char c;
    const int semaphore_num = 2;
    HANDLE semaphores[semaphore_num] = {FinishSignal,Empty};

    while (WaitForMultipleObjects(semaphore_num, semaphores, FALSE,
                                  INFINITE) == WAIT_OBJECT_0 + 1) {
        /* Funkcija za unos karaktera sa tastature.*/
        c = getch();

        /* Ukoliko je unet karakter q ili Q signalizira se programskim
           nitima zavrsetak rada; Vrednost semafora se inkrementira za
           2 da bi signaliziralo obema nitima.*/
        if(c =='q' || c =='Q')
            ReleaseSemaphore(FinishSignal,2,NULL);

        /* Ulazak u kriticnu sekciju B kako bi se obezbedio
           sinhronizovan pristup deljenoj promenljivoj (kruznom baferu
           ring).
        */
        EnterCriticalSection(&BufferAccess);
            ringBufPutChar(&ring, c);

        /* Napustanje kriticne sekcije.*/
        LeaveCriticalSection(&BufferAccess);

        /* V(F) - operacija uvecava semafor F za jedan.*/
        ReleaseSemaphore(Full,1,NULL);
    }
    return 0;
}
```

Potrošač

Programska nit potrošač, čeka ili na semaforu *Full* ako je njegova vrednost jednka nuli (bafer prazan)) ili signalizaciju za završetak rada niti (*FinishSignal*). Ukoliko bafer nije prazan (*Full* različito od nule), potrošač pokušava da uđe u kritičnu sekciju *BufferAccess*, pomoću funkcije *EnterCriticalSection*. Ukoliko programska nit proizvođač nije prethodno zauzela kritičnu sekciju *BufferAccess*, potrošač je zauzima i čita podatak iz bafera i ispisuje ga na konzolu. Nakon čitanja podatka iz bafera potrošač povećava vrednost semafora *Empty* za jedan.

```
/* Funkcija programske niti potrosaca.*/
DWORD WINAPI consumer(LPVOID param) {
    char c;
    printf("Znakovi preuzeti iz kruznog bafera:\n");

    const int semaphore_num = 2;
    HANDLE semaphores[semaphore_num] = {FinishSignal,Full};

    // P(F) - operacija  smanjuje senmafor F za jedan ukoliko je to
              moguce ;
    // Ako je F=0 nit ceka ReleaseSemaphore(F);
    while (WaitForMultipleObjects(semaphore_num, semaphores, FALSE,
                                  INFINITE) == WAIT_OBJECT_0 + 1) {
        // Ulazak u kriticnu sekciju B
        EnterCriticalSection(&BufferAccess);

        // Citanje kruznog bafera;
        c = ringBufGetChar(&ring);

        // Napustanje kriticne sekcije;
        LeaveCriticalSection(&BufferAccess);

        // Ispis na konzolu;
        printf("%c",c);

        // V(E) - operacija uvecava semafor E za jedan;
        ReleaseSemaphore(Empty,1,NULL);

        // Cekanje da bi se ilustrovalo trajanje obrade.
        Sleep(SLEEPING_TIME);
    }

    return 0;
}
```

NAPOMENA:

1. Za svaki program potrebno je obezbediti regularan završetak. To podrazumeva da se svaka nit završi izlaskom iz funkcije tela niti, a ne korišćenjem funkcije *exit*. Reakcija na signal završetka se može realizovati npr. korišćenjem funkcije *WaitForMultipleObjects*.

2. Pre završetak programa, potrebno je sačekati da se sve stvorene niti završe, korišćenjem funkcije *WaitForSingleObject*.

3. Obratiti pažnju da se svi alocirani resursi (kontekst niti, semafori, kritične sekcije, tajmeri) oslobode pre završetka programa.

Zadatak

Realizovati program koji konvertuje mala slova u velika slova. Formirati tri programske niti:

- ulazna programska nit,
- programska nit obrade i
- izlazna programska nit.

Ulazna programska nit prihvata ulazni karakter i smešta ga u ulazni kružni niz maksimalne veličine *RING_SIZE*. Programska nit obrade, preuzima karaktere iz ulaznog niza i vrši njihovo konvertovanje u velika slova. Konvertovani karakter se zatim smešta u izlazni kružni niz. Izlazna programska nit preuzima karaktere iz izlaznog kružnog niza i ispisuje ih na standardni izlaz.

Pretvaranje malih u velika slova se vrši oduzimanjem vrednosti karaktera sa 0x20. Prilikom realizacije voditi se primerom sa vežbi.