*Област вежби: Конкурентно програмирање*

# СИНХРОНИЗАЦИЈА И СИГНАЛИЗАЦИЈА ПРОГРАМСКИХ НИТИ

## Предуслови:

- Rpi2 рачунар (нису потреби додатаци),
- Преводиоц *GCC* освежен на верзију 4.7 или новију,
- Подешен мрежни приступ на један од начина представљених у документу „УВОД - Raspberry Pi рачунар“ уколико се ради преко мреже. Ако се Rpi2 рачунар користи као самосталан рачунар овај захтев се може занемарити,
- Познавање језика Це и материјала из вежбе „УВОД У КОНКУРЕНТНО ПРОГРАМИРАЊЕ.

## Семафори

Семафори су још један од механизама конкурентног програмирања. Користе се за синхронизацију извршавања нити у оквиру једног процеса.

Нека је С ненегативан цео број, над којим су дефинисане две операције – УВЕЋАЈ(С) и УМАЊИ(С). Када је је број С већи од нуле (С > 0) семафор је сигнализиран. Када С има вредност нула (С = 0) семафор није сигнализиран. Вредност С не може бити мања од нуле.

Операција УВЕЋАЈ(С): променљива С се увећава за један једном недељивом операцијом. Недељивост се обезбеђује на системском нивоу.

Операција УМАЊИ(С): умањење вредности С за један једном недељивом (енг. *atomic*) операцијом. Умањење се врши уколико је то могуће (С > 0). Ако је С=0, умањење није могуће. У пракси семафор се реализује као структура која дефинише С и ред чекања на семафору.

Функције које се користе у раду са POSIX семафорима су:

```
int sem_init (sem_t* sem, int pshared, unsigned int value);
```

| **Функција** | **Опис** |
|---|---|
| *sem_init* | Ствара нови семафор. |
| **Параметри** | **Опис** |
| *Sem* | Показивач на објекат семафора који се иницијализује. |
| *Pshared* | Вредност која означава да ли се семафор користи унутар једног процеса или га дели више процеса. Увек 0 у нашим примерима. |
| *Value* | Представља почетну вредност семафора. |
| **Повратна вредност** | **Опис** |
| *Int* | Стварање семафора је успело ако је повратна вредност једнака нули. |

```
int sem_wait (sem_t* sem);
```

| **Функција** | **Опис** |
|---|---|
| *sem_wait* | Функција преводи нит која је позива у стање чекања на сигнализацију од стране семафора.<br>Нит ће чекати на овом месту све док семафор има вредност 0, тј. док је у несигнализираном стању. Када се сигнализација догоди, нит се поново покреће, а вредност семафора умањује за 1. |
| **Параметри** | **Опис** |
| *sem* | Показивач на објекат који представља семафор на чију сигнализацију нит треба да чека. |
| **Повратна вредност** | **Опис** |
| *int* | Уколико је функција успешно извршена (нит је сачекала сигнализацију семафор) ова функција враћа нулу. У супротном, повратна вредност ће бити -1. |

```
int sem_post (sem_t* sem);
```

| **Функција** | **Опис** |
|---|---|
| *sem_post* | Сигнализира семафор, то јест увећава вредност семафора за 1. |
| **Параметри** | **Опис** |
| sem | Показивач на објекат семафора који се сигнализира. |
| **Повратна вредност** | **Опис** |
| *int* | Уколико је семафор успешно сигнализиран функција враћа вредност 0. У супротном, повратна вредност ће бити -1. |

## Пример 1 (vezba_2a)

Програмска нит која чека на унос са тастатуре и за сваки унети карактер повећава бројач (*counter*) за један. Уносом тастера 'q' или 'Q' програм завршава са радом.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

static sem_t semaphore;

static int counter;

char getch(void);

/* Telo niti. */
void* counter_thread (void *pParam)
{
    while (1)
    {
        /* Cekaj na signal da je pritisnut taster. */
        sem_wait(&semaphore);

        /* Povecaj brojac i ispisi vrednost. */
        counter++;
        printf("\r counter = %d\n", counter);
        fflush(stdout);
    }
}

int main (void)
{
    pthread_t hThread;

    sem_init(&semaphore, 0, 0);
    pthread_create(&hThread, NULL, counter_thread, 0);

    /* Glavna petlja programa, zavrsava se pritiskom na 'q'. */
    while (1)
    {
        /* Preuzmi znak. Ako je preuzeti znak 'q', zavrsi program. */
        if (getch() == 'q')
        {
             break;
        }

        /* Obavesti nit da je pritisnut taster. */
        sem_post(&semaphore);
    }

    sem_destroy(&semaphore);

    return 0;
}
```

**НАПОМЕНЕ:**

**1. Приликом стварања семафора обратити пажњу на његову почетну вредност. Ако је почетна вредност већа од нуле треба имати у виду да је тај семафор већ сигнализиран и да ће нитима које чекају на његов сигнал у старту бити омогућен даљи рад.**

**2. Увећавање вредности семафора (функција *`sem_post`*) није могуће преко максималне вредности која је дефинисана симболом SEM_VALUE_MAX у заглављу *semaphore.h*. Сви позиви *`sem_post`* који би довели до премашивања максималне вредности, игноришу се.**

## Прикључивање нити

Ово поглавље започиње једним примером:

Узмимо најпре у обзир чињеницу да извшравање програма престаје са престанком извршавања његове главне нити (*main* нит). То значи да када би унутар главне нити покренули неку функцију у новој нити и одмах затим наишли на крај функције main, извшрење целог програма би се прекинуло пре него што би функција новонастале нити имала прилику да се заврши. Због тога је у овом случају потребно да главна нит на неки начин сачека на свршетак извршавања новонастале нити па тек онда да и она сама заврши са радом. Другим речима, овај случај је један од примера у којима је потребно да се главна и новонастала нит синхронизују. Овакав модел синхронизације, у којем једна нит чека на завршетак друге, се у POSIX терминологији назива **прикључивање нит (енг. join)**. На овај начин, ток програма који се извршавао у нити која се чека прикључује се току програма нити која чека, и каже се да се једна нит прикључује другој.

Ако поново погледамо примере из Вежбе 1 видећемо да у сваком од њих постоји потреба за поменутим моделом синхронизације. У свима њима се на крају функције *main* налази позив функције *getchar* и на тај начин се одговорност синхронизације пребацује на корисника, тј. претпоставља се да ће корисник унети знак тек када све нити завше са радом. Међутим, у оквиру POSIX стандарда постоји механизам који регулише овакву синхронизацију нити:

```
int pthread_join (pthread_t thread, void** value);
```

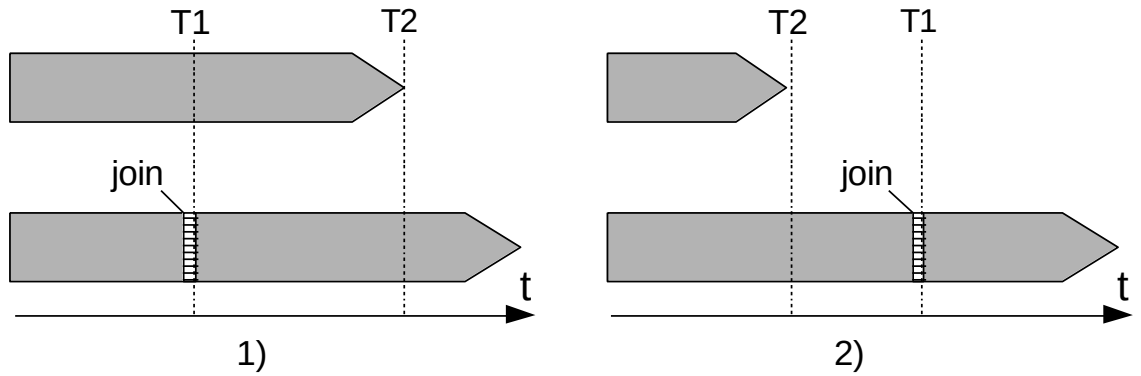
| **Функција** | **Опис** |
|---|---|
| *pthread_join* | Функција за прикључивање нити. Позива се из нити којој се прикључује, а аргумент је нит која се прикључује. |
| **Параметри** | **Опис** |
| *thread* | Променљива која представља нит која се прикључује, тј. на чији завршетак се чека. |
| *value* | Показивач на меморијску локацију у којој ће бити смештена повратна вредност нит (коју нити враћа имплицитно, или непосредним позивом функције *pthread_exit*). Вредност показивача може бити и NULL. Тада повратна вредност нити неће бити сачувана. |
| **Повратна вредност** | **Опис** |
| *int* | Повратна вредност је 0 уколико је нит успешно прикључена. У супротном, повратна вредност је код грешке. |

## Пример 2 (vezba_2aa)

Следећи пример илуструје коришћење функције *pthread_join*. Нити чекају једна другу у круг, а семафорима се обезбеђује да не дође до случаја да нит покуша са чекањем нити која још увек није покренута.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

static sem_t startSemaphore1;
static sem_t startSemaphore2;
static sem_t startSemaphore3;

static pthread_t hThread1;
static pthread_t hThread2;
static pthread_t hThread3;

void* thread1 (void *pParam)
{
    sem_wait(&startSemaphore1);

    /* Prikljuci se na nit 2. */
    pthread_join(hThread2, NULL);

    printf("Pozdrav iz niti 1.\n");

    return 0;
}

void* thread2 (void *pParam)
{
    sem_wait(&startSemaphore2);

    /* Prikljuci se na nit 3. */
```

```
    pthread_join(hThread3, NULL);

    printf("Pozdrav iz niti 2.\n");

    return 0;
}

void* thread3 (void *pParam)
{
    sem_wait(&startSemaphore3);

    printf("Pozdrav iz niti 3.\n");

    return 0;
}

int main (void)
{
    sem_init(&startSemaphore1, 0, 0);
    sem_init(&startSemaphore2, 0, 0);
    sem_init(&startSemaphore3, 0, 0);

    pthread_create(&hThread1, NULL, thread1, 0);
    pthread_create(&hThread2, NULL, thread2, 0);
    pthread_create(&hThread3, NULL, thread3, 0);

    /* Signaliziranje nitima da mogu da pocnu sa radom.
    Ovim se osigurava da nijedna nit ne dodje do funkcije pthread_join
    a da nit koju zeli da prikljuci nije pokrenuta.
    */
    sem_post(&startSemaphore1);
    sem_post(&startSemaphore2);
    sem_post(&startSemaphore3);

    /* Prikljuci se na nit 1. */
    pthread_join(hThread1, NULL);

    sem_destroy(&startSemaphore1);
    sem_destroy(&startSemaphore2);
    sem_destroy(&startSemaphore3);

    return 0;
}
```




На слици су илустрована два случаја чекања нити на прикључење. У првом случају нит којој се прикључује прелази у стање чекања (тренутак T1) пре него што је нит која се прикључује завршила свој ток (тренутак T2). У том смислу нит којој се прикључује заиста чека одређени временски период док се нит која се прикључује не заврши. Међутим, у другом случају нит која се прикључује завршава са радом (тренутак T2) пре него што је нит која прикључује спремна да је прикључи (тренутак T2). Због тога нит која се прикључује, иако је завршила са радом, мора остати у меморији све док је нит која је прикључује не прикључи и покупи њену повратну вредност. Тек тада ће сви ресурси које је нит заузела бити ослобођени. Због тога се прикључивање нити сматра и врстом ослобађања ресурса. Дакле, да би се обезбедио регуларан завршетак програма потребно је прикључити све нити (осим главне) које се направе током извршавања програма.

**НАПОМЕНЕ:**

**1. Приликом стварања, нит може бити направљена као „неприкључива“, и у том случају њени ресурси се ослобађају одмах кад се њено извршавање заврши. Због тога се функција *pthread_join* не може употребити за „неприкључиву“ нит.**

**2. Нити се подразумевано стварају као „прикључиве“.**

## Пример 3 (vezba_2b)

У примеру 3 приказано је решење задатка из Вежбе 1 применом семафора. У главној функцији *main* стварају се три нити које извршавају исту функцију *print*. Свака од нити чека сигнализацију свог семафора. При стварању семафора почетна вредност првог семафора у низу је један. Тиме је он иницијализован у сигнализираном стању и због тога прва нит креће одмах да се извршава.

За разлику од решења са критичним секцијама, где свака нит приступа дељеној променљивој, проверава њену вредност и у зависности од ње исписује свој број, синхронизација у овом примеру је обезбеђена семафорима. Нит нема потребе

да стално испитује услов и на тај начин окупира процесорско време, већ је систем обавештава када може да отпочне са извршавањем.

**НАПОМЕНА:**

**Семафори обезбеђују механизам сигнализације између нити, а објекти искључивог приступа недељив приступ дељеном ресурсу.**

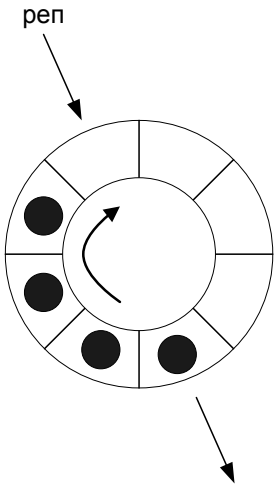
```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t semaphores[3];

void* print (void *pParam)
{
    int n = (int)pParam;
    int i;

    for (i = 0; i < 1000; i++)
    {
        /* Svaka nit ceka signalizaciju na svom semaforu i
        signalizira semafor sledecoj niti. */
        sem_wait(&semaphores[n]);
        printf("%d", n+1);
        fflush(stdout);
        /* Signaliziraj narednu nit. */
        sem_post(&semaphores[(n + 1) % 3]);
    }
    return 0;
}

int main (void)
{
    pthread_t hPrint1;
    pthread_t hPrint2;
    pthread_t hPrint3;

    /* Formiranje semafora za niti;
    Svakoj niti je namenjen jedan semafor;
    Semafor prve niti je inicijalno postavljen
    na jedinicu da bi niti otpocele sa radom. */
    sem_init(&semaphores[0], 0, 1);
    sem_init(&semaphores[1], 0, 0);
    sem_init(&semaphores[2], 0, 0);

    /* Formiranje niti;
    Sve niti imaju isto telo funkcije. */
    pthread_create(&hPrint3, NULL, print, (void*)2);
    pthread_create(&hPrint2, NULL, print, (void*)1);
    pthread_create(&hPrint1, NULL, print, (void*)0);

    pthread_join(hPrint3, NULL);
```

```
    pthread_join(hPrint2, NULL);
    pthread_join(hPrint1, NULL);

    sem_destroy(&semaphores[0]);
    sem_destroy(&semaphores[1]);
    sem_destroy(&semaphores[2]);

    return 0;
}
```

## Пример 4 (vezba_2c) Произвођач – Потрошач

Однос произвођач-потрошач предсатвља један од честих случајева синхронизације две програмске нити.

Претпоставимо да имамо кружни бафер (енг. *circular buffer*) са два показивача, репом (енг. *tail*) и главом (енг. *head*), где показивач реп показује на прву слободну позицију у баферу у коју се може сместити податак, а показивач глава указују на позицију са које се чита ускладиштени податак. Нека су две програмске нити (произвођач и потрошач) везане преко исте меморијске локације, у овом случају кружног бафера. Прва програмска нит произвођач (енг. producer) прозводи податке и уноси их у бафер, док их потрошач (енг. consumer) чита из бафера и користи. Пошто две различите програмске нити приступају истом меморијском ресурсу (кружни бафер) потребно је синхронизовати њихов приступ том дељеном ресурсу. Синхронизација се успоставља увођењем објекта искључивог приступа који штити критичну секцију за операције везане за рад са дељеним ресурсом. Такође, програмска нит произвођач треба да се блокира уколико је бафер пун, односно програмска нит потрошач треба да се блокира уколико је бафер празан.




Кружни бафер
- *реп* позиција на коју се уписује
- *глава* позиција са које се чита

ПРОИЗВОЂАЧ

- производи податак
- чека, уколко је бафер пун
- улази у критичну секцију
- уноси податак у бафер на позицију *реп*
- излази из критичне секције

ПОТРОШАЧ

- чека уколико је бафер празан
- улази у критичну секцију
- чита податак из бафера са позиције *глава*
- излази из критичне секције

## Реализација

### Основна програмска нит *main*

У основној програмској нити main формирају се два семафора: *semEmpty* і *semFull*. Почетна вредност *semEmpty* семафора је *RING_SIZE* и указује на број слободних места у кружном баферу. Семафор *semFull* се формира са почетном вредношћу нула и указује на број попуњених места у кружном баферу. Формира се и трећи семафор, *semFinishSignal*, који служи да се нитима сигнализира када треба да прекину са радом. У *main* програмској нити покрећу се две нове програмске нити: потрошач и произвођач. Програмска нит *main* након покретања програмских нити одлази у стање чекања док се новонастале програмске нити не заврше.

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

    /* Formiranje semEmpty, semFull i semFinishSignal semafora. */
    sem_init(&semEmpty, 0, RING_SIZE);
    sem_init(&semFull, 0, 0);
    sem_init(&semFinishSignal, 0, 0);

    /* Inicijalizacija objekta iskljucivog pristupa. */
```

```
    pthread_mutex_init(&bufferAccess, NULL);

    /* Formiranje programskih niti: proizodjac i potrosac. */
    pthread_create(&hProducer, NULL, producer, 0);
    pthread_create(&hConsumer, NULL, consumer, 0);

    /* Cekanje na zavrsetak formiranih niti. */
    pthread_join(hConsumer, NULL);
    pthread_join(hProducer, NULL);

    /* Oslobadjanje resursa. */
    sem_destroy(&semEmpty);
    sem_destroy(&semFull);
    sem_destroy(&semFinishSignal);
    pthread_mutex_destroy(&bufferAccess);

    printf("\n");

    return 0;
}
```

## Кружни бафер

Кружни бафер је претстављен структуром *RingBuffer*.

Та структура се састоји од следећих елемената:

1. *data* - низ ASCII знакова, величине *RING_SIZE*
2. *tail* - представља позицију на коју се смешта нови податак у низ *data*
3. *head* - позиција са које се чита податак из низа data

Операције за рад са кружним бафером дефинисане су као функције:

- ringBufGetChar(r) - операција за читање податка из низа са позиције *head*
- ringBufPutChar(r, c) - операција за смештање податка у низ на позицију *tail*

Приликом приступа подацима (ASCII znaci) у низу *data*, индекси *tail* и *head* се деле по модулу са величином низа (*RING_SIZE*) како би индекси били у границама величине низа.

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

/* Operacije za rad sa kruznim baferom. */
char ringBufGetChar (struct RingBuffer *apBuffer)
{
    int index;
```

```
    index = apBuffer->head;
    apBuffer->head = (apBuffer->head + 1) % RING_SIZE;
    return apBuffer->data[index];
}

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

## Потрошач

За програмску нит потрошач битна су два сигнала: *semFull* и *semFinishSignal*. У случају да је семафор *semFull* сигнализиран, нит потрошач треба да прочита знак из кружног бафера и сигнализира семафор *semEmpty*. У случају да је семафор *semFinishSignal* сигнализиран нит потрошач треба да прекине своје извршење. Дакле, нит потрошач мора да чека на сигнал два семафора. Због тога блокирајућа природа функције *sem_wait* није погодна, јер ако нит крене да чека један семафор неће моћи да одреагује на сигнализацију другог. Зато се уместо те функције користи се неблокирајућа функција: sem_trywait.

```
int sem_trywait (sem_t* sem);
```

| **Функција** | **Опис** |
|---|---|
| *sem_trywait* | Функција која проверава у ком стању се налази семафор. |
| **Параметри** | **Опис** |
| *sem* | Показивач на објекат семафора који се проверава. |
| **Повратна вредност** | **Опис** |
| *int* | Уколико је семафор у сигнализираном стању ова функција враћа 0. У супротном, повратна вредност је -1. Код грешке се тада налази у променљивој *errno*. Вредност променљиве *errno* је EAGAIN ако је семафор у несигнализираном стању. |

```
/* Funkcija programske niti potrosaca.*/
void* consumer (void *param)
{
    char c;

    printf("Znakovi preuzeti iz kruznog bafera:\n");

    while (1)
    {
        if (sem_trywait(&semFinishSignal) == 0)
        {
            break;
        }

        if (sem_trywait(&semFull) == 0)
```

```
        {
            /* Pristup kruznom baferu. */
            pthread_mutex_lock(&bufferAccess);
            c = ringBufGetChar(&ring);
            pthread_mutex_unlock(&bufferAccess);

            /* Ispis na konzolu. */
            printf("%c", c);
            fflush(stdout);

            /* Cekanje da bi se ilustrovalo trajanje obrade. */
            usleep(SLEEPING_TIME);

            sem_post(&semEmpty);
        }
    }

    return 0;
}
```

## Произвођач

Програмска нит произвођач чека два догађаја: или да потрошач преузме податак из бафера и повећа *semEmpty* семафор за један, уколико је семафор *semEmpty*=0 (кружни бафер пун), или сигнализацију за завршетак рада нити (*semFinishSignal*). Почетна вредост семафора *semEmpty* је једнака величини бафера *RING_SIZE*. Уколико кружни бафер није пун (*semEmpty* различит од нуле), произвођач преузима карактер са тастатуре и покушава да заузме објекат искључивог приступа *bufferAccess* функцијом *pthread_mutex_lock*. Уколико је нека друга програмска нит, у овом случају потрошач, заузела објекат искључивог приступа, нит прелази у стање чекања све док програмска нит потрошач не ослободи објекат искључивог приступа функцијом *pthread_mutex_unlock*. Објекат искључивог приступа синхронизује рад програмских нити са дељеним ресурсом (кружним бафером). Након уношења ASCII знака у бафер потрошач повећава семафор *Full* за један.

Уколико је притиснути знак ‘q’ или 'Q' програмска нит произвођач сигнализира обема нитима да заврше са радом, повећавајући вредност семафора *FinishSignal* за два.

```
/* Funkcija programske niti proizvodjaca. */
void* producer (void *param)
{
    char c;

    while (1)
    {
        if (sem_trywait(&semFinishSignal) == 0)
        {
            break;
```

```
        }

        if (sem_trywait(&semEmpty) == 0)
        {

            /* Funkcija za unos karaktera sa tastature. */
            c = getch();

            /* Ukoliko je unet karakter q ili Q signalizira se
               programskim nitima zavrsetak rada. */
            if (c == 'q' || c == 'Q')
            {
                /* Zaustavljanje niti; Semafor se oslobadja 2 puta
                da bi signaliziralo obema nitima.*/
                sem_post(&semFinishSignal);
                sem_post(&semFinishSignal);
            }

            /* Pristup kruznom baferu. */
            pthread_mutex_lock(&bufferAccess);
            ringBufPutChar(&ring, c);
            pthread_mutex_unlock(&bufferAccess);

            sem_post(&semFull);
        }
    }

    return 0;
}
```

## Задатак

Реализовати програм који претвара мала слова у велика слова. Формирати три програмске нити:

1. улазна програмска нит,
2. програмска нит обраде и
3. излазна програмска нит.

Улазна програмска нит прихвата улазни карактер и смешта га у улазни кружни низ максималне величине *RING_SIZE*. Програмска нит обраде преузима карактере из улазног низа и претвара их у велика слова. Претворени карактер се затим смешта у излазни кружни низ. Излазна програмска нит преузима карактере из излазног кружног низа и исписује их на стандардни излаз.

Претварање малих у велика слова се обавље одузимањем константе 0x20 од вредности карактера. Приликом реализације водити се примером са вежби.