*Област вежби: Конкурентно програмирање*

# РУКОВАЊЕ ВРЕМЕНСКИ КОНТРОЛИСАНИМ ДОГАЂАЈИМА И ПОНИШТАВАЊЕ 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 вежби „УВОД У КОНКУРЕНТНО ПРОГРАМИРАЊЕ“ и „СИНХРОНИЗАЦИЈА И СИГНАЛИЗАЦИЈА ПРОГРАМСКИХ НИТИ“.

## Увод

У овој вежби пример произвођач-потрошач из Вежбе 2 је проширен следећим додатним захтевима:

1. Уколико у кружни бафер 5 секунди није уписан ни један знак, програм треба да заврши са радом.
2. Поред програмске нити потрошача потребно је формирати и програмску нит која сваке 2 секунде исписује садржај целог кружног бафера.

Ова два захтева уводе нове проблеме.

## Потрошач

Програмска нит потрошач у овом примеру проверава оба услова за завршетак рада програма (корисник је унео знако ‘q’/’Q’, и пет секунди није било нових уписа у кружни бафер). Зато у њој није потребно чекати никакав знак за прекид из других нити.

Основу потрошача чини безусловна петља. Унутар петље нит проверава стање бафера. Уколико бафер није празан (семафор *semFull* је сигнализиран), потрошач покушава да уђе у критичну секцију кода. Ако произвођач није претходно закључао објекат искључивог приступа *bufferAccess*, потрошач га закључава и чита податак из бафера. Након изласка из критичне секције, потрошач проверава да ли је

прочитани знак ‘q’/’Q’ и у зависности од тога искаче из петље, или исписује знак на конзолу и сигнализира семафор *semEmpty*.

На крају петље нит се успављује на период од једне секунде и бројач *local_counter* се умањује за 1. Са друге стране, вредност бројача *local_counter* се на почетку поставља на 5 и поново враћа на ту вредност при сваком читању из бафера. То значи да ће бројач *local_counter* имати вредност 0 онда када нит потрошач пет пута одспава по једну секунду, а при том кружни бафер остане стално празан. Додавањем провере тог услова на почетак нити испуњава се први додатни захтев ове вежбе.

Такође, пре завршетка, потребно је да нит потрошач сигнализира семафор *semFinishSignal*.

Анализирати коректност оваквог решења у односу на постављени захтев:

- Да ли ће се програм увек прекинути после тачно 5 секунди неуношења знакова? Од чега то зависи?
- Да ли ће се прецизност поправити смањењем периода спавања и одговарајућим повећањем почетне вредности бројача *local_counter*? Зашто?

```
void usleep (unsigned long microseconds);
```

| **Функција** | **Опис** |
|---|---|
| *usleep* | Успављује нит на одређени број микросекунди. |
| **Параметри** | **Опис** |
| *microseconds* | Број који изражава на колико микросекунди нит треба да буде успавана. |

```
/* Nit potrosaca. */
void* consumer (void *param)
{
    char c;
    int local_counter = 5;

    while (1)
    {

        if (local_counter == 0)
        {
            break;
        }

        if (sem_trywait(&semFull) == 0)
        {
            /* Postavljanje brojaca na pocetnu vrednost. */
            local_counter = 5;
```

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

            /* Ukoliko je unet karakter q ili Q nit treba da se zavrsi.
*/
            if (c == 'q' || c == 'Q')
            {
                break;
            }

            printf("Consumer: %c\n",c);
            fflush(stdout);
            sem_post(&semEmpty);
        }

        /* Umanjenje brojaca koje oznacava da je protekla 1 sekunda. */
        local_counter -= 1;
        /* Uspavljivanje niti na jednu sekundu. */
        usleep(1000000);
    }

    sem_post(&semFinishSignal);

    return 0;
}
```

## Програмска нит за испис садржаја кружног бафера

Програмска нит за испис садржаја бафера наизменично проверава два семафора: *semFinishSignal* и *semPrintSignal*. Нит потрошач сигнализира семафор *semFinishSignal* када и она сама заврши са радом, чиме јавља овој нити да и она треба да заврши са радом. Насупрот тога, сигнализација *semPrintSignal* семафора значи да нит треба да испише садржај целог кружног бафера.

Други захтев ове вежбе тражи да се садржај бафера исписује сваке две секунде. То значи да семафор *semPrintSignal* треба сигнализирати тим темпом. Да би се тако нешто обезбедило могуће је применити механизам примењен у нити потрошач. Међутим, то решење има неколико мана, од којих се истичу две:

- За време док спава, нит не ради ништа друго, и
- Функција нити се оптерећује додатним кодом.

У том смислу, боље решење претставља употреба временски контролисаних догађаја, или скраћено, временске контроле.

У датотеци **timer_event.c** налази се једна имплементација руковаоца временски контролисаним догађајима. Додавањем те датотеке у пројекат, као и

укључивањем заглавља **timer_event.h** у датотеку са изворним кодом, програмеру се омогућава коришћење механизма временски контролисаних догађаја које руковаоц нуди.

Спрега руковаоца са корисником остварује се кроз две функције:

```
int timer_event_set (timer_event_t* timer_var, unsigned int delay, void*
(*routine)(void*), void* arg, int te_kind);
```

| **Функција** | **Опис** |
|---|---|
| *timer_event_set* | Активира временску контролу у руковаоцу. |
| **Параметри** | **Опис** |
| *timer_var* | Адреса промењиве која представља временску контролу. |
| *Delay* | Период временске контроле, изражен у милисекундама. |
| *Routine* | Адреса функције која ће бити извршена по истеку времена дефинисаног параметром *delay*. |
| *Arg* | Аргумент који ће бити прослеђен функцији. |
| *te_kind* | Дефинише тип временске контроле. Могуће су следеће вредности:<br>TE_KIND_ONCE – Временска контрола се покреће само једном.<br>TE_KIND_REPETITIVE – Временска контрола се покреће периодично. |
| **Повратна вредност** | **Опис** |
| *Int* | Ако је позив успешан, повратна вредност је 0. У супротном, враћа се код грешке. |

```
int timer_event_kill (timer_event_t timer_var);
```

| **Функција** | **Опис** |
|---|---|
| *timer_event_kill* | Зауставља временску контролу |
| **Параметри** | **Опис** |
| *timer_var* | Променљива која представља временску контролу коју треба зауставити. Ова променљива се постаља при позиву функције *timer_event_set*. |
| **Повратна вредност** | **Опис** |
| *Int* | Ако је позив успешан, повратна вредност је 0. У супротном, враћа се код грешке. |

Помоћу периодичне временске контроле сваких 2000ms (2 секунде) позива се функција *print_state_timer* која сигнализира семафор *semPrintSignal*. Када је семафор *semPrintSignal* сигнализиран, програмска нит за испис исписује садржај кружног бафера, водећи рачуна о приступу дељеном ресурсу.

```
/* Nit za ispis vrednosti karaktera iz kruznog bafera. */
void* print_state (void *param)
{
    int i;

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

        if (sem_trywait(&semPrintSignal) == 0)
        {
            /* Pristup kruznom baferu. */
            printf("------------\n");
            pthread_mutex_lock(&bufferAccess);
            for (i = 0 ; i < RING_SIZE ; i++)
            {
                printf("%c/", ring.data[i]);
            }
            printf("\n");
            pthread_mutex_unlock(&bufferAccess);
            printf("------------\n");
            fflush(stdout);
        }
    }
    return 0;
}

/* Funkcija vremenske kontrole koje se poziva na svake dve sekunde. */
void* print_state_timer (void *param)
{
    sem_post(&semPrintSignal);

    return 0;
}

void main (void)
{
...
    /* Formiranje vremenske kontrole za funkciju print_state_timer. */
    timer_event_set(&hPrintStateTimer,   2000,   print_state_timer,   0,
TE_KIND_REPETITIVE);
...
    /* Zaustavljanje vremenske kontrole. */
    timer_event_kill(hPrintStateTimer);
...
}
```

## Произвођач

Иако је улога нити произвођач у овом случају сведена на преузимање знака са тастатуре и његово уписивање у кружни бафер, јавља се проблем завршетка те нити. Проблем представља блокирајућа природа функције за преузимање знака са тастатуре. У Вежби 2 нит произвођач је сама сигнализирала семафор *semFinishSignal* (када прикупи знак 'q'/'Q') и онда одмах затим реаговала на тај семафор. Зато је у том примеру овај проблем остао прикривен. Међутим, овога пута не само да је провера да ли је унет знак 'q'/'Q' премештена у нит потрошач, него је уведен и још један услов за завршетак програма (неуношење ниједног знака 5 секунди). Поставља се питање како сигнализирати нити да прекине са радом ако она чека на унос знака.

Једно решење је коришћење механизма за поништавање, то јест прекидање рада, нити, који се нуди у **PThread** библиотеци. Основу тог механизма чини функција *pthread_cancel*:

```
int pthread_cancel (pthread_t thread);
```

| **Функција** | **Опис** |
|---|---|
| *pthread_cancel* | Функција сигнализира одређеној нити да прекине са радом, то јест поништава је. |
| **Параметри** | **Опис** |
| *thread* | Променљива која представља нит која се поништава. |
| **Повратна вредност** | **Опис** |
| *int* | Ако је позив успешан, повратна вредност је 0. У супротном, враћа се код грешке. |

Свака нит може бити поништива и непоништива. Подразумевано, све нити се стварају као поништиве.

Свака поништива нит може бити једног од два типа:

1. Поништива са задршком (енг. *deffered*), то јест синхроно поништива, и
2. Асинхроно поништива.

Асинхроно поништиву нити је могуће поништити у било ком тренутку. Насупрот томе, уколико је нит поништива са задршком онда ће сигнал за поништење чекати прву прилику када нит дође у тачку поништивости. На пример, тачке поништивости су неке од функција чекања, а тачка поништивости се може и експлицитно дефинисати позивом функције *pthread_testcancel*.

Подразумевано, све нити су поништиве са задршком. Тип поништивости нити се мења следећом функцијом:

```
int pthread_setcanceltype (int type, int* oldtype);
```

| **Функција** | **Опис** |
|---|---|
| *pthread_setcanceltype* | Мења тип поништивости нити која је позива. |
| **Параметри** | **Опис** |
| *type* | Идентификатор типа поништивости који ће бити постављен. Може бити једна од две вредности:<br>PTHREAD_CANCEL_DEFERRED – Са задршком (синхроно), и<br>PTHREAD_CANCEL_ASYNCHRONOUS – Асинхроно |
| *oldtype* | Показивач на променљиву у коју ће бити смештен идентификатор дотадашњег типа поништивости. |
| **Повратна вредност** | **Опис** |
| *int* | Ако је позив успешан, повратна вредност је 0. У супротном, враћа се код грешке. |

Тип поништивости нити може се мењати током њеног извршавања.

Коришћењем наведених функција могуће је прекинути извршавање нити произвођач, без обзира што користи блокирајућу функцију. На овај начин не само да се може регуларно прекинути њено извршавање, него се и функција нити растерећује кода који би водио рачуна о семафорима, или неким другим сигналима.

У наставку је дат код функције нити произвођач.

**Питање:** Зашто тип поништивости нити није само једном, на почетку нити, постављен да буде асинхрон? Какав би се ту проблем јавио?

```
/* Nit proizvodjaca. */
void* producer (void *param)
{
    char c;

    while (1)
    {
        sem_wait(&semEmpty);

        /* Promena tipa ponistivosti niti proizvodjaca. */
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
        c = getch();
        pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);

        pthread_mutex_lock(&bufferAccess);
        ringBufPutChar(&ring, c);
        pthread_mutex_unlock(&bufferAccess);
```

```
        sem_post(&semFull);
    }

    return 0;
}
```

## Задатак

1a). Реализовати први захтев ове вежбе коришћењем временске контроле.

Додати временску контролу која смањује глобалну промењиву *counter* сваке секунде (*counter* је иницијално 5). Када *counter* постане 0 завршити са радом програма. Промењива *counter* се поставља на 5 сваки пут када потрошач прочита знак из бафера. Из кода програмске нити потрошач избацити функцију за спавање.

Приликом релизације водити рачуна о синхронизованом приступу дељеној промењивој *counter*.

Анализирати коректност оваквог решења у односу на постављени захтев:

- Да ли ће се програм увек прекинути после тачно 5 секунди неуношења знакова? Од чега то зависи?
- Да ли ће се прецизност поправити смањењем периода временске контроле и одговарајућим повећањем почетне вредности бројача *counter*? Зашто?

Упоредити ове одговоре са одговорима датим у поглављу „Потрошач“.

1b). Променити код тако да се и нит за испис садржаја кружног бафера прекида поништавањем (из кода потпуно избацити семафор *semFinishSignal*). Поништавање обавити искључиво из нити главног програма.