Област вежби: Конкурентно програмирање

УВОД У КОНКУРЕНТНО ПРОГРАМИРАЊЕ

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 корити као самосталан рачунар овај захтев се може занемарити,
- Познавање језика Це.

Увод

Вишепроцесни (енг. *multitasking*) оперативни систем омогућава реализацију програма са више програмских процеса који могу да се извршавају конкурентно. У зависности од узрока који доводи до смене текућег процеса системи се могу поделити на две основне групе:

- **Системи без истискивања процеса (*non-preemtive*)**

  Код ове врсте вишепроцесних система до смене текућег процеса може доћи само уколико то захтева сам текући процес. Због тога је програмирање процеса отежано, пошто процеси морају повремено сами позивати распоређивач процеса да би омогућили смену процеса. Због ове особине овакви системи се често називају и кооперативни системи. Ова врста имплементације вишепроцесног окружења је примењена код оперативног система Windows 3.x.

- **Системи са истискивањем процеса (*preemptive*)**

  Ово су вишепроцесни системи код којих може доћи до смене текућег процеса и уколико то сам процес не тражи. Овакви системи додељују временски интервал (енг. *time slice*) за сваки процес. Након истека временског интервала ОС прекида текући процес, пребацујући га у стање приправан (енг. *ready*), и распоређује приправан процес највишег приоритета. Временски одсечак зависи од оперативног система и од процесора (приближна вредност на ОС Windows 3.x је 20ms). Због веома кратког временског интервала изгледа као да се више процеса извршава у исто време. Овакви системи се често називају и "прави" вишепроцесни системи. Пример њихове примене

су оперативни системи Unix, Linux, Windows 95/98 и Windows NT/2000/XP.

Један посао (енг. *job*) се састоји од једног или више процеса. Сваки процес може садржати једну или више програмских нити. Нити представљају основну јединицу оперативног система које конкуришу за време процесора. Сваки процес се покреће извршавањем једне нити која се назива главна програмска нит (у програмским језицима C/C++/Java извршење те нити започиње у функцији *main*). У току извршавања процеса, могуће је створити произвољан број нити из било које претходно створене нити. Сваки процес поседује ресурсе потребне за његово извршавање (додељени меморијски простор и друге системске ресурсе). Програмске нити деле ресурсе процеса којем припадају. Програм се извршава када распоређивач процеса преда управљање једној од нити процеса који припада програму.

Стварање програмских нити

Функција `pthread_create` ствара нову нит процеса:

```
int pthread_create (pthread_t* thread, const pthread_attr_t* attr,
                    void* (*start_routine)(void*), void* arg );
```

При стварању нове нити је потребно навести адресу одакле ће почети извршавање направљене нити. То је обично адреса неке функције дефинисане у програму. Програм може имати више нити које истовремено извршавају код исте функције.

НАПОМЕНА: Свака нит заузима засебан меморијски простор за податке.

Програмској нити је могуће проследити показивач на било који тип. У случају да нема параметара које треба проследити, показивач се поставља на NULL.

Функција	**Опис**
pthread_create	Ствара нову програмску нит
Параметри	**Опис**
thread	У ову променљиву се у случају успешног стварања нити смешта њен идентификациони број (енг. ID). Ова променљива се касније користи за референцирање нити.
attr	Показивач на структуру која представља атрибуте нити. У случају да је показивач NULL користе се подразумевани атрибути. То ће бити случај у већини примера на овом курсу.
start_routine	Адреса функције нити. Функција се декларише на следећи начин: **`void* ThreadRoutine (void* param);`**
arg	Показивач који ће бити прослеђен нити приликом стварања. Преко њега нит може приступити подацима које јој је програмер намењио.
Повратна вредност	**Опис**
int	Стварање нити је успело ако је повратна вредност 0. У супротном, повратна вредност је код грешке која је спречила стварање нити.

НАПОМЕНА:

1. Редослед стварања нити функцијом *`pthread_create`* не мора бити исти са редоследом покретања (створених) нити од стране оперативног система.

2. И *main* је нит која се конкурентно извршава са осталим нитима у систему. Завршетком функције *main*, завршава се програм.

Пример 1 (изворни код се налази у директоријуму vezba_1a_v1)

Програм са три програмске нити које исписују свој идентификатор на стандардни излаз. Треба приметити да смењивање програмских нити врши сам оперативни систем. Свака нит се извршава одређени временски одсечак (енг. *time slice*).

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

/* Telo niti 1. */
void* print1 (void *pParam)
{
    int i;

    for (i = 0 ; i < 1000 ; i++)
    {
        printf("1");
        fflush(stdout);
    }
```

```
    return 0;
}

/* Telo  niti 2. */
void* print2 (void *pParam)
{
    int i;

    for (i = 0 ; i < 1000 ; i++)
    {
        printf("2");
        fflush(stdout);
    }

    return 0;
}

/* Telo niti 3. */
void* print3 (void *pParam)
{
    int i;

    for (i = 0 ; i < 1000 ; i++)
    {
        printf("3");
        fflush(stdout);
    }

    return 0;
}

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

    /* Formiranje niti. */
    pthread_create(&hPrint1, NULL, print1, 0);
    pthread_create(&hPrint2, NULL, print2, 0);
    pthread_create(&hPrint3, NULL, print3, 0);

    getchar();

    return 0;
}
```

Претходни пример се може имплементирати и само једном функцијом (а не три као у претходном примеру), којој се прослеђује параметер приликом стварања нити. Она тај параметер преузима приликом стартовања и исписује на стандардни излаз. На основу једне дефиниције тела нити (једне функције) стварају се три нити, које имају засебне локалне променљиве, а све могу приступати заједничким глобалним променљивама. Изворни код примера се налази у директоријуму vezba_1a_v2.

НАПОМЕНА: При писању програма са више нити, ако је могуће, треба тежити да се користи једна функција као тело нити, а разлике надоместити параметризацијом, тј. прослеђивањем параметара приликом стварања нити.

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

/* Telo niti. */
void* print (void *pParam)
{
    char c = *(char*)pParam;
    int i;

    for(i = 0; i < 1000; i++)
    {
        printf("%c", c);
        fflush(stdout);
    }

    return 0;
}

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

    /* Argumenti niti. */
    char c1 = '1';
    char c2 = '2';
    char c3 = '3';

    pthread_create(&hPrint1, NULL, print, (void*)&c1);
    pthread_create(&hPrint2, NULL, print, (void*)&c2);
    pthread_create(&hPrint3, NULL, print, (void*)&c3);

    getchar();

    return 0;
}
```

Следећи пример (vezba_1a_v3) указује на типичну грешку приликом прослеђивања параметара нити.

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

void* print (void *pParam)
{
    char c = *(char*)pParam;
    int i;
```

```
    for(i = 0; i < 1000; i++)
    {
        printf("%c", c);
        fflush(stdout);
    }

    return 0;
}

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

    char c;
    /* Ovako ne treba raditi. */
    c = '1';
    pthread_create(&hPrint1, NULL, print, (void*)&c);
    c = '2';
    pthread_create(&hPrint2, NULL, print, (void*)&c);
    c = '3';
    pthread_create(&hPrint3, NULL, print, (void*)&c);

    getchar();

    return 0;
}
```

Горе наведени пример илуструје погрешан начин прослеђивања параметара. Разлог томе је што се користи само један примерак стандардног типа (*int, char*, итд.) или структуре чија вредност се прослеђује нити, а параметар се не преноси по вредности него по адреси.

У горе наведеном примеру стварају се три нити, а пре стварања дате нити, у промењиву **c** се уписује одговарајућа вредност параметра. Параметар се преноси преко адресе, што значи да ће све нити добити показивач на исти објекат, тј. исту адресу као улазни параметар.

Формирање нити се састоји из више корака: резервише се меморијски простор за нит, резервише се контекст нити, и на крају се покреће нит. Параметри се најчешће ишчитавају на почетку тела нити (функције), нпр. са:

```
char c = *(char*) lpParam;
```

До тог момента може доћи до прекључивања нити, у току припреме за формирање нити. Уколико нит *main* опет постане активна, она ће променити вредност локалне промењиве **c** пре но што ју је формирана нит прочитала.

У најгорем случају, нит *main* ће покренути стварање свих нити, и још увек ће бити активна. То доводи до тога да када створене нити постану активне, **c** ће имати

вредност 3, и свака нит ће исписивати исту вредност. С обзиром да прекључивање зависи од момента покретања и од оперативног система, можда је потребно покренути пример више пута да би се постигло нежељено понашање.

НАПОМЕНА: Приликом прослеђивања параметара нитима користити засебне примерке структура или стандардних типова, за сваку створену нит понаособ.

Објекат искључивог приступа

Објекти искључивог приступа (енг. mutex = **mut**ual **ex**clusion) представљају механизам за синхронизацију нити и штићење критичних секција кода. На пример, ако се некој структури података приступа из различитих нити са намером измене података у структури, ради се о критичној секцији те се мора обезбедити да не дође до истовременог приступа од стране различитих нити. Другим речима операција измене података прве нити мора се завршити да би друга нит добила право да мења или чита те податке. Овај механизам није неопходан једино у случају када обе нити само читају податке из дељене структуре, али када постоји и упис, мора се користити механизам штићења.

Дељени ресурс се штити објектом искључивог приступа тако што се тај објекат пре уласка нити у критичну секцију заузима, а при излазску из критичне секције ослобађа. По заузимању објекта искључивог приступа од стране једне нити ни једна друга нит га не може заузети докле год је нит која га је заузела не ослободи. Нити приступају дељеном ресурсу по редоследу заузимања објекта искључивог приступа.

Функције које се користе у раду са објектом искључивог приступа су:

```
int pthread_mutex_init (pthread_mutex_t* mutex,
                        pthread_mutexattr_t* attr);
```

Функција	**Опис**
pthread_mutex_init	Иницијализује објекат искључивог приступа.
Параметри	**Опис**
Mutex	Показивач на објекат искључивог приступа. Функција подразумева да је меморија за тај објекат већ заузета од стране корисника, то јест, да је објекат пре тога дефинисан.
Attr	Показивач на структуру која представља атрибуте објекта искључивог приступа. У случају да је показивач NULL користе се подразумевани атрибути. То ће бити случај у већини примера на овом курсу
Повратна вредност	**Опис**
Int	Повратна вредност је 0 у случају да је објекат искључивог приступа успешно иницијализован. У супротном, повратна вредност представља код грешке.

```
int pthread_mutex_lock (pthread_mutex_t* mutex);
```

Функција	**Опис**
pthread_mutex_lock	Заузимање објекта искључивог приступа.
Параметри	**Опис**
mutex	Показивач на објекат искључивог приступа. Функција подразумева да је објекат искључивог приступа претходно био иницијализован.
Повратна вредност	**Опис**
int	Повратна вредност је 0 у случају да је објекат искључивог приступа успешно заузет. У супротном, повратна вредност представља код грешке.

```
int pthread_mutex_unlock (pthread_mutex_t* mutex);
```

Функција	**Опис**
pthread_mutex_unlock	Ослобађање објекта искључивог приступа.
Параметри	**Опис**
mutex	Показивач на објекат искључивог приступа. Функција подразумева да је објекат искључивог приступа претходно био иницијализован.
Повратна вредност	**Опис**
int	Повратна вредност је 0 у случају да је објекат искључивог приступа успешно ослобођен. У супротном, повратна вредност претставља код грешке.

```
int pthread_mutex_destroy (pthread_mutex_t* mutex);
```

Функција	**Опис**
pthread_mutex_destroy	Супротна акција од иницијализације објекта искључивог проступа. Објекат се након позива ове функције више не може користити за регулисање искључивог приступа, све док не буде поново иницијализован.
Параметри	**Опис**
mutex	Показивач на објекат искључивог приступа.
Повратна вредност	**Опис**
int	Повратна вредност је 0 у случају да је објекат искључивог приступа успешно деиницијализован. У супротном, повратна вредност представља код грешке.

НАПОМЕНА:

1. Број заузимања објекта искључивог приступа означен је са N, а број ослобађања са M.

Ситуација	**Опис**
N = M	**Објекат искључивог приступа је правилно ослобођен**
N > M	**Објекат искључивог пристуап је и даље заузет**
N < M	**Незаузети објекат искључивог приступа је ослобођен. То може довести до нежељених последица, јер је понашање програма у том случају недефинисано.**

2. Када се нит заврши, објекат искључивог приступа мора бити ослобођен. У противном, може доћи до заустављања рада осталих нити ако оне чекају на ослобађање тог објекта искључивог приступа (енг. dead-lock).

3. Не користити *return* пре напуштања критичне секције заштићене објектом искључивог приступа.

4. Бити опрезан приликом коришћења функције `pthread_mutex_unlock` **у оквиру неког условног гранања (if, else) због могућности њеног неизвршавања (у зависности од услова) и самим тим довођења других нити, а и самог програма, у блокирано стање (енг. dead-lock).**

Пример 2 (vezba_1b_v1)

Програм са три програмске нити које исписују три различите реченице на стандардни излаз. Постоје две верзије примера: у првој (vezba_1b_v1) се не користе објекти искључивог приступа и исписи реченица ће се преклапати, што је нежељена ситуација. До преклапања долази јер програмске нити користе исти ресурс (стандардни излаз) без међусобне синхронизације. Односно, уколико се извршавање

прогама прекључи на следећу нит пре него што је претходна нит завршила са исписом своје реченице, долази до мешања исписа.

Да не би дошло до преклапања, критична секција нити се штити објектом искључивог приступа (vezba_1b_v1) везаног за стандарни излаз (испис на екрану), где свака програмска нит чека ослобађање објекта искључивог приступа уколико је он заузет.

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

static pthread_mutex_t cs_mutex;


void* print1 (void *pParam)
{
    int i;
    for (i = 0 ; i < 100 ; i++)
    {
        pthread_mutex_lock(&cs_mutex);

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

        pthread_mutex_unlock(&cs_mutex);
    }

    return 0;
}

void* print2 (void *lpParam)
{
    int i;
    for (i = 0 ; i < 100 ; i++)
    {
        pthread_mutex_lock(&cs_mutex);

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

        pthread_mutex_unlock(&cs_mutex);
    }

    return 0;
}

void* print3 (void *lpParam)
```

```
{
    int i;
    for(i = 0 ; i < 100 ; i++)
    {
        pthread_mutex_lock(&cs_mutex);

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

        pthread_mutex_unlock(&cs_mutex);
    }

    return 0;
}

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

    pthread_mutex_init(&cs_mutex, NULL);

    pthread_create(&hPrint1, NULL, print1, 0);
    pthread_create(&hPrint2, NULL, print2, 0);
    pthread_create(&hPrint3, NULL, print3, 0);

    getchar();

    pthread_mutex_destroy(&cs_mutex);

    return 0;
}
```

Задатак

Модификовати приложени пример (vezba_1a_v1) тако да се створе 3 нити на основу једне функције, при чему свака програмска нит има свој идентификатор (1, 2 и 3).

Помоћу објеката искључивог приступа и дељених променљивих, реализовати механизам смењивања нити тако да се на екрану исписује

123123123123123123123123123…….

то јест, да се прво активира нит 1, па нит 2 а затим нит 3, 100 пута. После 100 исписа, нити се завршавају. Механизам за смењивање нити треба да обезбеди да нит након свог исписа сигнализира наредној нити да може да испише свој идентификатор.

Задатак реализовати помоћу објекта искључивог приступа и дељене променљиве. Пошто приступ дељеној променљивој представља критичну секцију нити, потребно ју је заштитити коришћењем објекта искључивог приступа. Све три програмске нити реализовати истом функцијом којој се прослеђује идентификатор програмске нити.

Уколико се између два приступа критичној секцији уметне пауза (помоћу функције Sleep), програм ће се брже завршити. Зашто?