Լաբորատոր աշխատանք 6

Հոսքեր (Threads)

Ինչպես և պրոցեսները, հոսքերը հնարավորություն են տալիս իրականացնել մի քանի գործողություններ մրցակցային կերպով: Մեկ պրոցեսը կարող է պարունակել մի քանի հոսք: Պրոցեսի բոլոր հոսքերը միմյանցից անկախ կատարում են միևնույն ծրագիրը և կիսում են գլոբալ հիշողության միևնույն տարածքը:

Հոսքի ստեղծումը

Նոր հոսքի ստեղծումն իրականացվում է pthread_create() ֆունկցիայի կիրառմամբ:

```
#include <pthread.h>

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

Նոր հոսքը սկսում է կատարվել **start** ֆունկցիայով, որին փոխանցվում է **arg** արգումենտը՝ **start(arg)**: Այն հոսքը, որից տեղի է ունեցել pthread_create() կանչը, շարունակում է աշխատանքը՝ կատարելով հաջորդ հրամանը: Ֆունկցիայի մյուս 2 արգումենտներն ունեն հետևյալ նշանակությունը.

- **thread** արգումենտը *pthread_t* տիպի ցուցիչ է, որի մեջ պատճենվում է հոսքի id-ն: Այն կարող է օգտագործվել հետագա կանչերի ընթացքում:
- **attr** արգումենտը *pthrerad_attr_t* տիպի ցուցիչ է, որը սահմանում է ստեղծվող հոսքի տարբեր ատրիբուտներ: Եթե *attr* արգումենտը սահմանվում է որպես NULL, ապա հոսքը ստեղծվում է ատրիբուտների լռելյայն արժեքներով:

Հոսքն ավարտում է աշխատանքը, երբ start ֆունկցիան իրականացնում է return հրամանը կամ կանչում է pthread_exit() ֆունկցիան:

```
#include <pthread.h>
void pthread_exit(void *retval);
```

retval արգումենտը սահմանում է հոսքի վերադարձվող արժեքը:

Ավարտված հոսքի հետ միավորումը

pthread_join() ֆունկցիան սպասում է thread ID-ով սահմանված հոսքի ավարտին: Եթե հոսքն արդեն ավարտվել է, ապա ֆունկցիան անմիջապես ավարտվում է: Այս գործողությունն անվանում են միավորում:

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **retval);
```

Եթե *retval*-ը ոչ *NULL* ցուցիչ է, ապա այն ստանում է ավարտված հոսքի վերադարձվող արժեքը, այսինքն՝ այն արժեքը, որը սահմանվել է, երբ հոսքը արժեք է վերադարձրել կամ կանչել է *pthread_exit()*:

Simple_thread ծրագիրը

simple_thread ծրագիրը գլխավոր հոսքում (main() ֆունկցիայիում) pthread_create() ֆունկցիայի կիրառմամբ ստեղծում է նոր հոսք: Նոր ստեղծված հոսքն աշխատանքը սկսում է **threadFunc** ֆունկցիայից, որին որպես արգումենտ փոխանցվում է "Hello world" տողը: Այն պարզապես տպում է որպես արգումենտ փոխանցված տողը և վերադարձնում է այդ տողի երկարությունը:

Գլխավոր հոսքից իրականացվում է նոր ստեղծված հոսքի միավորում ՝ pthread_join() ֆունկցիայի կիրառմամբ: Դա նշանակում է, որ այդ կետում գլխավոր հոսքի աշխատանքն արգելափակվում է այնքան ժամանակ, մինչև մյուս հոսքն ավարտի աշխատանքը: pthread_join() կանչի արդյունքում գլխավոր հոսքը ստանում է երկրորդ հոսքի վերադարձրած արժեքը և արտածում այն:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc simple_thread.c -pthread -o simple
- Կատարել ծրագիրը.
  ./simple

Check_thread ծրագիրը

Check_thread ծրագրում ներկայացվում է **pthread_equal()** և **pthread_self()** ֆունկցիաների կիրառումը: **pthread_equal()** ֆունկցիան հնարավորություն է տալիս ստուգել, թե արդյոք 2 հոսքերի ID-ները նույնն են: **pthread_self()** ֆունկցիան վերադարձնում է տվյալ հոսքի id-ն: *pthread_equal()* ֆունկցիան անհրաժեշտ է, քանի որ *pthread_t* տվյալների տիպը պետք է դիտարկվի որպես տվյալների անթափանց տիպ (opaque data type): Այսինքն՝ անհրաժեշտ է խուսափել այդ տվյալների հետ ուղղակի աշխատանքից (օրինակ՝ t1 == pthread_self()) և կիրառել Pthreads API-ի տրամադրած ֆունկցիաները:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc check_thread.c -pthread -o check
- Կատարել ծրագիրը.
  ./check

Detached_thread ծրագիրը

detached_thread ծրագրում ներկայացվում է անջատված (detached) հոսքի ստեղծումը: Անջատված անվանում են այն հոսքը, որն ավարտվելիս ավտոմատ կերպով հեռացվում է հիշողությունից՝ առանց pthread_join() կանչի: Հոսքը կարելի է նշել որպես անջատված՝ կանչելով pthread_detach() ֆունկցիան:

```
#include <pthread.h>
int pthread_detach(pthread_t thread);
```

Հոսքը կարող է ինքն իրեն անջատել հետևյալ կերպ.

```
pthread_detach(pthread_self());
```

Հոսքը կարելի է նշել որպես անջատված նաև ստեղծման պահին, այլ ոչ հաջորդիվ pthread_detach() կանչի միջոցով: Դա կատարվում է հերթի ատրիբուտների ստրուկտուրայի վրա pthread_attr_setdetachstate() ֆունկցիայի կիրառմամբ:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc detached_thread.c -pthread -o detached
- Կատարել ծրագիրը.
  ./detached

Հոսքերի սինխրոնացում
Thread_increment և thread_increment_mutex ծրագրերը

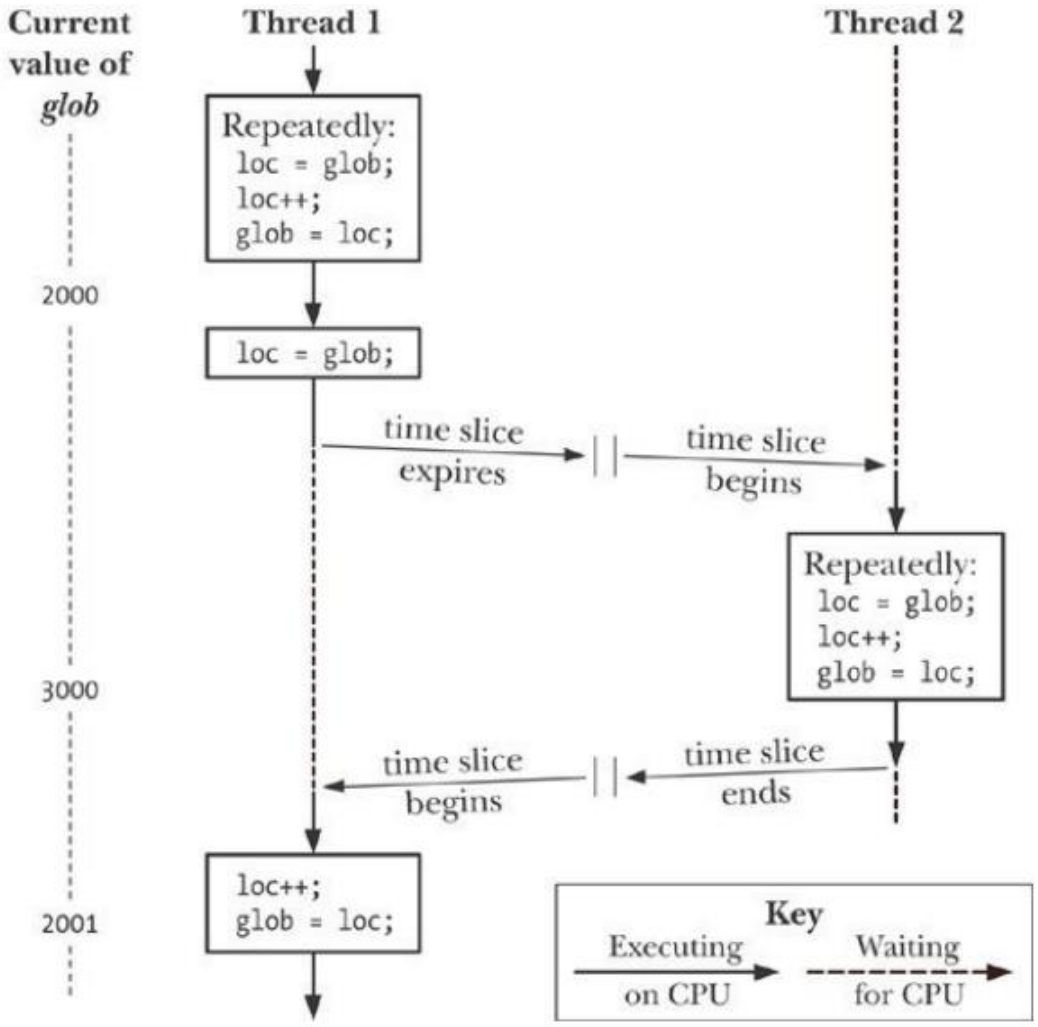
thread_incr ծրագիրը ստեղծում է 2 հոսք, որոնցից յուրաքանչյուրը կատարում է նույն ֆունկցիան: Ֆունկցիան կատարում է ցիկլ, որի ամեն իտերացիայում գլոբալ փոփոխականի արժեքը մեծացնում է 1-ով: Իտերացիաների քանակը որոշվում է հրամանային տողի արգումենտով: Եթե հրամանային տողից արգումենտ չի փոխանցվում, ապա կիրառվում է լռելյայն արժեք (10.000.000): Վերջում գլխավոր հոսքն արտածում է գլոբալ փոփոխականի արժեքը:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc thread_increment.c -pthread -o increment
- Կատարել ծրագիրը.
  ./increment

Եթե ծրագիրը կատարենք 1000 իտերացիայի համար, ապա սպասվող վերջնարդյունքը կլինի 2000, ինչը և արտածվում է ծրագրի կողմից:
./thread_increment 1000
glob = 2000

10.000.000 իտերացիայի դեպքում սպասվող արժեքը կլինի 20.000.000, սակայն այս դեպքում ծրագիրն արտածում է անկանխատեսելի արժեքներ.
./thread_increment
glob = 10971837



Նկ. 1 Գլոբալ փոփոխականի արժեքը մեծացնող 2 հոսքեր,
որոնք չեն սինխրոնացվում

Այս իրավիճակից խուսափելու նպատակով անհրաժեշտ է կիրառել հոսքերի սինխրոնացման մեխանիզմ՝ մյուտեքս (MUTEX – MUTual EXclusion): Մյուտեքսները հնարավորություն են տալիս համաձայնեցնել ընդհանուր ռեսուրսի օգտագործումն այնպես, որ մի հոսքը չփորձի կարդալ / փոփոխել ընդհանուր փոփոխականն այն պահին, երբ մյուսը փոփոխում / կարդում է այն:

Մյուտեքսը *pthread_mutex_t* տիպի փոփոխական է: Մինչ օգտագործելը, այն միշտ պետք է սկզբնավորել.

pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

Սկզբնավորումից հետո մյուտեքսը ազատ վիճակում է: Զբաղեցնելու և ազատելու համար կիրառվում են **pthread_mutex_lock()** և **pthread_mutex_unlock()** ֆունկցիաները:

thread_increment_mutex ծրագրում ներկայացված է thread_increment ծրագրի նոր տարբերակը, որում գլոբալ փոփոխականի հետ աշխատանքի ժամանակ կիրառվում է մյուտեքս:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc thread_increment_mutex.c -pthread -o increment_mutex
- Կատարել ծրագիրը.
  ./increment_mutex

Պայմանական փոփոխականներ, thread_cond_variables ծրագիրը

Thread_cond_variables ծրագրում գլխավոր հոսքը աշխատանքը շարունակելու համար սպասում է available փոփոխականի արժեքի ավելացմանը, ինչը տեղի է ունենում 2-րդ հոսքում: Դա նշանակում է, որ գլխավոր հոսքը պետք է սպասի 2-րդ հոսքի ավարտին՝ կանչելով pthread_join() ֆունկցիան: 2-րդ հոսքը available փոփոխականի արժեքն ավելացնելուց հետո դեռ 2 վայրկյան աշխատում է: Գլխավոր հոսքն իր հերթին սպասման վիճակից դուրս գալուց հետո աշխատում է 3 վայրկյան:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc thread_cond_variables.c -pthread -o cond_variables
- Կատարել ծրագիրը.
  ./cond_variables

Ստացվում է, որ ծրագրի ընդհանուր աշխատանքի ժամանակը 5 վայրկյան է, մինչդեռ գլխավոր հոսքը 2 վայրկյան վատնում է՝ սպասելով 2-րդ հոսքի ավարտին: Այս դեպքում օգտակար կլիներ կիրառել ազդանշան ուղարկելու / ստանալու որևէ մեխանիզմ, այնպես, որ 2-րդ հոսքում available փոփոխականի ավելացումից անմիջապես հետո

ազդանշան ուղարկվեր գլխավոր հոսքին՝ հաղորդելով փոփոխության մասին, իսկ գլխավոր հոսքը սպասեր ազդանշան ստանալուն, այլ ոչ հոսքի աշխատանքի ավարտին: Դա կատարվում է պայմանական փոփոխականների միջոցով:

Պայմանական փոփոխականը հոսքին հնարավորություն է տալիս մյուս հոսքերին տեղեկացնել ընդհանուր փոփոխականի արժեքի փոփոխության մասին, և թույլ է տալիս մյուս հոսքերին սպասել այդպիսի իրադարձության:

Պայմանական փոփոխականն ունի **pthread_cond_t** տիպը: pthread_cond_signal() և pthread_cond_broadcast() ֆունկցիաները հաղորդում են, որ cond փոփոխականի արժեքը փոխվել է: pthread_cond_wait() ֆունկցիան արգելափակում է հոսքը մինչև նոր ազդանշանի հայտնվելը:

Հոսքային լոկալ հիշողություն, elements ծրագիրը

Ֆունկցիան համարվում է անվտանգ (thread-safe), եթե այն միաժամանակ կարող է կանչվել տարբեր հոսքերի կողմից: elements ծրագրում կիրառվող getElement() ֆունկցիան (գտնվում է get_element.h ֆայլում) անվտանգ չէ, քանի որ ֆունկցիան աշխատում է գլոբալ փոփոխականի հետ՝ առանց սինխրոնացման մեխանիզմ կիրառելու: Ֆունկցիան կարելի է անվտանգ դարձնել՝ օգտագործելով հոսքային լոկալ հիշողություն: Այդ նպատակով գլոբալ փոփոխականները հայտարարվում են **__thread** բանալի-բառի միջոցով:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc -pthread -o elements elements.c
- Կատարել ծրագիրը.
  ./elements

Հոսքի չեղարկումը, thread_cancel ծրագիրը

Հոսքի չեղարկումն իրականացվում է pthread_cancel() ֆունկցիայի միջոցով: thread_cancel ծրագիրը ստեղծում է նոր հոսք, որը կատարում է անվերջ ցիկլ: Ցիկլի իտերացիայում այն սպասում է 1 վայրկյան և տպում է հերթական արժեքը: Քանի որ այս հոսքը երբեք չի հասնի return հրամանին, ապա այն ավարտելու միակ միջոցը չեղարկելն է: Ծրագրի գլխավոր հոսքը սպասում է 3 վայրկյան և ուղարկում է չեղարկման հարցում:

Ծրագիրը կատարելու համար անհրաժեշտ քայլերն են.

- Ստեղծել ծրագրի կատարվող ֆայլը.
  gcc -pthread -o thread_cancel thread_cancel.c
- Կատարել ծրագիրը.
  ./thread_cancel

Առաջադրանքներ.

1. Ստեղծել նոր հոսք և այդ հոսքին փոխանցել կամայական տող: Նոր ստեղծված հոսքից վերադարձնել ստացված տողի երկարությունը: Գլխավոր հոսքում տպել ավարտված հոսքից վերադարձված արժեքը:

2. Կատարել check_thread ծրագիրը: Բացատրել pthread_equal() ֆունկցիայի և == օպերատորի տարբերությունը:

3. Ստեղծել նոր հոսք և այն դարձնել անջատված (detached)՝ կիրառելով pthread_detach() ֆունկցիան:

4. Կատարել thread_increment ծրագիրը, բացատրել ստացված անսպասելի արդյունքը: Փոփոխել ծրագիրն այնպես, որ գլխավոր հոսքի կողմից ստեղծված առաջին հոսքն ավատի աշխատանքը, որից հետո միայն երկրորդ հոսքը սկսի աշխատել: Բացատրել տարբերությունը thread_increment_mutex ծրագրի հետ:

5. Կատարել thread_cond_variables ծրագիրը: Փոփոխել ծրագիրն այնպես, որ գլխավոր հոսքը սպասի երկրորդ հոսքի կողմից ուղարկված պայմանական փոփոխականի ազդանշանին, այլ ոչ հոսքի ավարտին: Կրկին կատարել ծրագիրը և բացատրել ծրագրի կատարման ժամկետների տարբերությունը:

6. Կատարել elements ծրագիրը և բացատրել ցուցադրված հաղորդագրությունը: Ծրագիրը փոփոխել այնպես, որ այն օգտագործի հոսքային լոկալ հիշողություն և կրկին կատարել:

7. Կատարել thread_cancel ծրագիրը: Ծրագիրը փոփոխել այնպես, որ չեղարկման հարցում ստանալուց հետո հոսքը չչեղարկվի և շարունակի աշխատանքը:

8. Գրել ծրագիր, որը կստեղծի 2 հոսք: Ստեղծված առաջին հոսքին այն պետք է փոխանցի որպես հրամանային տողի արգումենտ ստացված թիվը: Առաջին հոսքը պետք է կրկնապատկի թիվը և վերադարձնի, որից հետո գլխավոր հոսքը այն պետք է փոխանցի 2-րդ հոսքին: 2-րդ հոսքը պետք է եռապատկի ստացված արժեքը և վերադարձնի: Գլխավոր հոսքում արտածել ստացված վերջնարդյունքը:

9. Գրել ծրագիր, որը գլոբալ int data[10] զանգվածը լրացնում է [1,10] միջակայքի թվերով: Ընդ որում, ծրագրի գլխավոր հոսքը պետք է լրացնի միայն կենտ տարրերը: Զույգ տարրերը պետք է լրացվեն գլխավոր հոսքից ստեղծված նոր հոսքում: Հոսքերը պետք է աշխատեն մրցակցային կերպով: Վերջում արտածել լրացված զանգվածի տարրերը և համոզվել դրանց իսկության մեջ:

10. Գրել ծրագիր, որը կհաշվի տրված data[10] զանգվածի տարրերի գումարը: Ընդ որում, զանգվածի առաջին 5 տարրերի գումարը և վերջին 5 տարրերի գումարը պետք է հաշվել առանձին հոսքերում՝ մրցակցային կերպով: