НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра обчислювальної техніки\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «Паралельне програмування»

(назва дисципліни)

на тему: «Розробка програмного забезпечення для паралельних комп’ютерних систем»

Студента 3 курсу \_\_\_ІП-42\_\_\_ групи

напряму підготовки 050103

«Програмна інженерія»

\_\_\_\_ Водотійця Д.І. \_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ- 2017 р.

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) інформатики та обчислювальної техніки

( повна назва )

Кафедра обчислювальної техніки

( повна назва )

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 «Програмна інженерія»

# (шифр і назва)

## ***З А В Д А Н Н Я***

### НА КУРСОВИЙ ПРОЕКТ СТУДЕНТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Водотійцю Денису Ігоровичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

керівник роботи Корочкін Олександр Володимирович к.т.н.**,** доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 11 травня 2017 р.

3. Вхідні дані до роботи (приклад)

- засоби роботи з процесами в мові Ада

- математична задача МА = МВ\*МС

- структури ПКС ОП та ПКС ЛП

- мови і бібліотеки програмування: Ада

- засоби організації взаємодії процесів: захищений модуль, механізм

рандеву мови Ада

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд засобів роботи з процесами в мові Ада

- розробка і тестування програми ПРГ1 для ПКС ОП

- розробка і тестування програми ПРГ2 для ПКС ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС ОП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

7. Дата видачі завдання \_\_\_\_\_\_\_\_хх\_\_\_\_\_

#### КАЛЕНДАРНИЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання розділу 1 | 23.02.2016 |
| 2 | Виконання розділу 2 | 23.03.2016 |
| 3 | Виконання розділу 3 | 23.04.2016 |
| 4 | Оформлення КР | 8.05.2016 |
| 5 | Перевірка КР викладачем | 11.05.2016 |
| 6 | Захист КР | 18.05.2016 |

**Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_**Водотієць Д. І.**\_\_**

( підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Корочкін О.В.**\_\_\_**

( підпис ) (прізвище та ініціали)

ЗМІСТ

ВСТУП…………………………………………………………………………… 3

РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПОТОКАМИ В МОВІ C#…..............5

* 1. Багатопотоковість ……………………………………….………….. 6
  2. Типи реалізації потоків .…………………………………………… 7
  3. Взаємодія потоків…………………………………………………… 7

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………………...

ВСТУП

Темою даної курсової роботи є «Розробка програмного забезпечення для паралельних комп’ютерних систем», що передбачає собою розробку програм для обчислення математичної задачі у паралельних комп’ютерних системах з загальною та локальною пам’яттю, а також розбір теоретичних відомостей за варіантом.

У першому розділі розглянуто засоби роботи з потоками в мові C #.

РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПОТОКАМИ В МОВІ C#

Листів

Лист

1

Літера

6.050103

ІП-42

**Розробка програмного забезпечення для паралельних комп’ютерних систем**

НТУУ КПІ 17 4103 - 000 ПЗ

Зм.

Виконав

Керівник

Конс.

Н. контр.

Зав. каф.

№ докум.

Водотієць Д.І.

Корочкін О.В.

Корочкін О.В.

Стіренко С.Г

Лит

Підпис

Дата

1

**1.1. Потік**

Потік[1] — в [інформатиці](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Інформатика) так називається спосіб програми розділити себе на дві чи більше паралельні задачі. Реалізація потоків та [процесів](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Процес (програмування)) відрізняються в різних [операційних системах](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Операційна система), але загалом потік міститься всередині процесу і різні потоки одного [процесу](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процес (інформатика)) спільно розподіляють деякі ресурси, в той час як різні [процеси](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процес (інформатика)) ресурси не розподіляють.

В системах з одним [процесором](https://uk.wikipedia.org/wiki/CPU" \o "CPU) багатопотоковість реалізується загалом поділом часу виконання («кванти часу»), дуже подібно до паралельного виконання багатьох задач: процесор послідовно переключається між різними потоками. Це переключення контексту відбувається настільки швидко, що у кінцевого користувача створюється ілюзія одночасного виконання. На [багатопроцесорних](https://uk.wikipedia.org/w/index.php?title=%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0&action=edit&redlink=1" \o "Багатопроцесорна система (ще не написана)) чи на [багатоядерних системах](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D1%8F%D0%B4%D0%B5%D1%80%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Багатоядерна система) робота потоків здійснюється справді одночасно, оскільки різні потоки і [процеси](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процес (інформатика)) виконуються буквально одночасно різними процесорами або ядрами процесора.

Багато сучасних операційних систем прямо підтримують квантування часу і багатопроцесорну роботу потоків через [планувальник процесів](https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D1%83%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%96%D0%B2" \o "Планувальник процесів). [Ядро операційної системи](https://uk.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8) дозволяє програмісту маніпулювати потоками через інтерфейс [системних викликів](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%B8%D0%B9_%D0%B2%D0%B8%D0%BA%D0%BB%D0%B8%D0%BA" \o "Системний виклик). Деякі реалізації викликають потоки ядра, оскільки [легковагові процеси](https://uk.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%B3%D0%BA%D0%BE%D0%B2%D0%B0%D0%B3%D0%BE%D0%B2%D1%96_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B8&action=edit&redlink=1" \o "Легковагові процеси (ще не написана)) ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) lightweight process, LWP) є спеціальним типом потоків ядра, що розподіляють деякі стани та інформацію.

Поза тим, програма може емулювати роботу потоків, використовуючи таймер, сигнали або інші методи, щоб перервати власне виконання і послідовно виконувати різні задачі власним квантуванням часу. Такий спосіб іноді зветься потоками користувацького простору ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) user-space threads) або [волокнами](https://uk.wikipedia.org/wiki/%D0%92%D0%BE%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

**1.2. Порівняння потоків з процесами**

Потоки відрізняються від традиційних [процесів](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D0%BA%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%96_%D0%BD%D0%B0%D1%83%D0%BA%D0%B8)" \o "Процес (комп'ютерні науки)) [багатозадачних](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D1%96%D1%81%D1%82%D1%8C" \o "Багатозадачність) [операційних систем](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Операційна система), в тому що [процеси](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процес (інформатика)):

* на загал, незалежні;
* дублюють значну частину інформації про [стан](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B8_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%83);
* мають окремий [адресний простір](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%B4%D1%80%D0%B5%D1%81%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%96%D1%80&action=edit&redlink=1" \o "Адресний простір (ще не написана));
* взаємодіють тільки через системні [міжпроцесорні механізми комунікацій](https://uk.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D1%94%D0%BC%D0%BE%D0%B4%D1%96%D1%8F_%D0%BC%D1%96%D0%B6_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B0%D0%BC%D0%B8" \o "Взаємодія між процесами).

Потоки всередині [процесу](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процес (інформатика)), з іншої сторони, розподіляють інформацію про стан [процесу](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), і прямо доступаються до спільної пам'яті та інших ресурсів. Переключення контексту між потоками процесу на загал швидше, ніж переключення контексту між процесами. Описуючи ситуацію такі системи, як [Windows NT](https://uk.wikipedia.org/wiki/Windows_NT" \o "Windows NT) та [OS/2](https://uk.wikipedia.org/wiki/OS/2), кажуть, що мають «дешеві» потоки та «дорогі» процеси; в інших операційних системах ситуація не дуже відмінна.

**1.3. Багатопотоковість**

Багатопотоковість[2] ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *multi-threading*),  — властивість [операційної системи](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) або [застосунку](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA), яка полягає в тому, що [процес](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), породжений в операційній системі, може складатися з кількох [нитей](https://uk.wikipedia.org/wiki/%D0%9D%D0%B8%D1%82%D1%8C), що виконуються паралельно, або навіть одночасно на багатопроцесорних системах. При виконанні деяких завдань таке розділення може досягти ефективнішого використання ресурсів [комп'ютера](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80).

Головною метою багатопотоковості є квазі-багатозадачність на рівні одного виконуваного процесу, тобто всі потоки виконуються в адресному просторі процесу. Окрім цього, всі потоки процесу мають не тільки спільний адресний простір, але і спільні дескриптори файлів. Процес, що виконується, має як мінімум однин (головний) потік.

Головні переваги в багатопотоковості:

* Спрощення програми в деяких випадках, за рахунок використання загального адресного простору;
* Менші відносно процесу часові витрати на створення ниті і взаємодію між нитями;
* Підвищення продуктивності процесу за рахунок розпаралелювання процесорних обчислень і операцій вводу/виводу.

**1.4. Взаємодія потоків**

У багатонитевому середовищі часто виникають проблеми, зв'язані з використанням паралельними виконуваними птоками одних і тих же даних або пристроїв. Для вирішення подібних проблем використовуються такі методи взаємодії потоків, як взаємовиключення ([м'ютекси](https://uk.wikipedia.org/wiki/%D0%9C%27%D1%8E%D1%82%D0%B5%D0%BA%D1%81" \o "М'ютекс)), [семафори](https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Семафор (програмування)), [критичні секції](https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D1%82%D0%B8%D1%87%D0%BD%D1%96_%D1%81%D0%B5%D0%BA%D1%86%D1%96%D1%97" \o "Критичні секції) і [події](https://uk.wikipedia.org/w/index.php?title=%D0%9F%D0%BE%D0%B4%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1" \o "Подія (програмування) (ще не написана)).

* Взаємовиключення (mutex, м'ютекс) — це об'єкт синхронізації, який встановлюється в особливий сигнальний стан, коли не зайнятий жодним потоком. Тільки один поток володіє цим об'єктом у будь-який момент часу, звідси і назва таких об'єктів (від англійського mutually exclusive access — взаємно виключний доступ) — одночасний доступ до загального ресурсу виключається. Після всіх необхідних дій м'ютекс звільняється потоком, надаючи іншим потокам доступ до загального ресурсу.
* Семафори - є доступні ресурси, які можуть займатися кількома потоками в один і той же час, поки обсяг ресурсів не спустіє. Тоді додаткові потоки повинні чекати, поки необхідна кількість ресурсів не буде знову доступна. Семафори дуже ефективні, оскільки вони дозволяють одночасний доступ до ресурсів.
* Події - корисні в тих випадках, коли необхідно послати повідомлення потоку, що відбулося певна подія. Наприклад, при асинхронних операціях вводу/виводу з одного пристрою, система встановлює подію в сигнальний стан, коли закінчується якась з цих операцій. Один поток може використовувати кілька різних подій в декількох операціях, що перекриваються, а потім чекати приходу сигналу від будь-якого з них.
* Критичні секції забезпечують синхронізацію подібно м'ютексам за винятком того, що об'єкти, що представляють критичні секції, доступні в межах одного процесу.

Події, м'ютекси і семафори також можна використовувати в однопроцесному застосунку, проте критичні секції забезпечують швидший і ефективніший механізм синхронізації взаємного виключення.

**1.5. Засоби роботи з потоками в мові С#**

C # підтримує паралельне виконання коду через багатопоточність.

Програма на C# запускається як єдиний потік, автоматично створюваний CLR(Common Language Runtime - загальномовне виконуюче середовище) і операційною системою ("головний" потік), і стає багатопотоковою за допомогою створення додаткових потоків. Ось простий приклад і його висновок:

class ThreadTest

{

static void Main()

{

Thread thread = new Thread(Print);

thread.Start(); // Виконати Print() у новому потоці

while (true)

{

Console.Write("Hi!"); // Друкувати в косоль «Hi!»

}

}

static void Print()

{

while (true)

{

Console.Write("Hello, world!");

// Друкувати в косоль «Hello, world!»

}

}

}

У головному потоці створюється новий потік thread, виконуючий метод, який безперервно друкує «Hi!». Одночасно головний потік безперервно друкує «Hello, world!». CLR призначає кожному потоку свій стек, так що локальні змінні зберігаються окремо.

**Управління багатопоточністю** здійснює планувальник потоків, цю функцію CLR зазвичай делегує операційній системі[4]. Планувальник потоків гарантує, що активним потокам виділяється відповідний час на виконання, а потоки, які очікують або блоковані, наприклад, на очікуванні ексклюзивної блокування, або користувача введення - не споживають часу CPU.

На однопроцесорних комп'ютерах планувальник потоків використовує квантування часу - швидке перемикання між виконанням кожного з активних потоків. Це призводить до непередбачуваного поведінки, як в першому прикладі, де кожна послідовність «Hi!» і «Hello, world!» відповідає кванту часу, виділеного потоку. У

На багатопроцесорних комп'ютерах багатопоточність реалізована як суміш квантування часу і справжнього паралелізму, коли різні потоки виконують код на різних CPU. Необхідність квантування часу все одно залишається, тому що операційна система повинна обслуговувати як свої власні потоки, так і потоки інших додатків.

Кажуть, що потік витісняється, коли його виконання призупиняється через зовнішніх факторів типу квантування часу. У більшості випадків потік не може контролювати, коли і де він буде витіснений.

Для **створення потоків** використовується конструктор класу Thread, який приймає як параметр делегат типу ThreadStart, який вказує метод, який потрібно виконати. Делегат ThreadStart визначається так:

*public delegate void ThreadStart();*

Виклик методу Start() починає виконання потоку. Потік триває до виходу з виконуваного методу. Ось приклад, який використовує повний синтаксис C# для створення делегата ThreadStart:

*class ThreadTest*

*{*

*static void Main(){*

*Thread t = new Thread(new ThreadStart(Run));*

*t.Start(); // Виконати Run() в новому потоці*

*Run (); // Одночасно запустити Run() у головному потоці*

*}*

*static void Run ()*

*{*

*Console.WriteLine("Hello, world!");*

*}*

У цьому прикладі потік виконує метод Run () одночасно з головним потоком. Результат - два майже одночасних «Hello, world!»

Потік має властивість IsAlive, що повертає true після виклику Start () і до завершення потоку.

Потік, який закінчив виконання, не може бути початий знову.

Потоку можна задати ім’я, використовуючи властивість Name. Це надає велику зручність при налагодженні: імена потоків можна вивести в Console.WriteLine() і побачити у вікні Debug - Threads в Microsoft Visual Studio. Ім'я потоку може бути призначено в будь-який момент, але тільки один раз - при спробі змінити його буде згенеровано виключення.

Головному потоку також можна призначити ім'я - в наступному прикладі доступ до головного потоку здійснюється через статичну властивість CurrentThread класу Thread:

*class ThreadNaming*

*{*

*static void Main()*

*{*

*Thread.CurrentThread.Name = "main";*

*Thread worker = new Thread(Go);*

*worker.Name = "worker";*

*worker.Start();*

*Go();*

*}*

*static void Go()*

*{*

*Console.WriteLine("Hello from " + Thread.CurrentThread.Name);*

*}*

*}*

Властивість Priority визначає, скільки часу на виконання буде виділено потоку щодо інших потоків того ж процесу. Існує 5 градацій пріоритету потоку:

enum ThreadPriority {Lowest, BelowNormal, Normal, AboveNormal, Highest}

Значення пріоритету стає істотним, коли одночасно виконуються декілька потоків.

Установка пріоритету потоку на максимум ще не означає роботу в реальному часі (real-time), так як існують ще пріоритет процесу додатки. Щоб працювати в реальному часі, потрібно використовувати клас Process з простору імен System.Diagnostics для підняття пріоритету процесу:

Process.GetCurrentProcess (). PriorityClass = ProcessPriorityClass.High;

Від ProcessPriorityClass.High один крок до найвищого пріоритету процесу - Realtime. Встановлюючи пріоритет процесу в Realtime, ви говорите операційній системі, що хочете, щоб ваш процес ніколи не витіснявся. Якщо ваша програма випадково потрапить в нескінченний цикл, операційна система може бути повністю заблокована. Врятувати вас в цьому випадку зможе тільки кнопка вимкнення живлення. З цієї причини ProcessPriorityClass.High вважається максимальним пріоритетом процесу, придатним до вживання.

Якщо real-time програма має користувальницький інтерфейс, може бути не бажано піднімати пріоритет його процесу, так як оновлення екрану буде з'їдати занадто багато часу CPU - гальмуючи весь комп'ютер, особливо якщо UI досить складний. Зменшення пріоритету головного потоку в поєднанні з підвищенням пріоритету процесу гарантує, що real-time потік не буде витіснятися перемальовуванням екрану, але не рятує від гальм весь комп'ютер, тому що операційна система все ще буде виділяти багато часу CPU всьому процесу в цілому. Ідеальне рішення полягає в тому, щоб тримати роботу в реальному часі і користувальницький інтерфейс в різних процесах (з різними пріоритетами), що підтримують зв'язок через Remoting або shared memory.

**1.6. Висновки до розділу 1**

У даному розділі розглядалася багатопотоковість в мові програмування C. Був виконаний аналіз багатопотоковості в даній мові програмування. Короткий огляд даної мови багатопотокового програмування можна виконати за такими основними параметрами:

1. Створення потоку. Первинний (головний) потік створюється автоматично , решта - за допомогою класу Thread, якому в якості параметрів передається делегат типу ThreadStart, який вказує метод, який потрібно виконати. Виклик методу Start починає виконання потоку.

2. Засоби синхронізації потоків. У C # для синхронізації використовуються: lock, Mutex, Semaphore, EventWaitHandle, Wait and Pulse, Interlocked, volatile (для безпечного не блокує доступу до полів).

В C# існує стандартний набір функцій для повноцінної роботи багатопоточних систем. При виборі конкретного засобу програмування потрібно керуватися специфічними завданнями, які необхідно вирішити в даній задачі.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Введение в многопоточность [Електронний ресурс] – Режим доступу: http://www.codenet.ru/progr/cpp/threads.php
2. Ars Technica article about multithreading, etc [Електронний ресурс] – Режим доступу: http://archive.arstechnica.com/paedia/images/m-hyperthreading-1.html
3. Багатопотоковість [Електронний ресурс] – Режим доступу: http://qps.ru/GWbLr
4. Работа с потоками в C# [Електронний ресурс] – Режим доступу: http://rsdn.ru/article/dotnet/CSThreading2.xml