Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

# Методичні вказівки

до виконання самостійних робіт
з дисципліни «Паралельні та розподілені обчислення»
для студентів спеціальності
123 «Комп’ютерна інженерія»
усіх форм навчання

Кривий Ріг
2021

Укладачі: Музика І. О., канд. техн. наук, доцент
Кузнєцов Д. І., канд. техн. наук, доцент
Рецензент: Жосан А. А., канд. техн. наук, доцент

Дані методичні вказівки містять завдання самостійної роботи з дисципліни «Паралельні та розподілені обчислення» для студентів спеціальності 123 «Комп’ютерна інженерія». Представлено основи створення додатків для паралельної та розподіленої обробки даних мовою Python із застосуванням семафорів, конвеєрних алгоритмів, іменованих каналів, механізму передачі повідомлень та мережних сокетів.

Розглянуто
на засіданні кафедри
комп’ютерних систем та мереж

Протокол № 1
від 27.08.2021 р.

Схвалено
на вченій раді факультету
інформаційних технологій

Протокол № 1
від 30.08.2021 р.

# ВСТУП

Паралельні та розподілені обчислення – спосіб організації роботи програми, що ґрунтується на використанні декількох процесорів чи обчислювальних ядер комп’ютерної системи. Сьогодні паралельні обчислення використовуються навіть при розробці програмного забезпечення для мобільних пристроїв. Технології побудови та відлагодження паралельних програм досить складні і, тому виходять за межі курсу звичайного програмування. Організацію програми з використанням декількох потоків підтримують майже всі сучасні мови програмування, зокрема: Java, C#, C++, Python, Object Pascal, Objective-C та ін.

Дисципліна «Паралельні та розподілені обчислення» – одна із фундаментальних дисциплін професійного спрямування у загальній системі підготовки фахівців за спеціальністю 123 «Комп'ютерна інженерія». Передумовою успішного вивчення даної дисципліни є володіння навичками програмування та алгоритмізації, які отримані під час вивчення дисциплін попередніх курсів навчання: «Основи інформаційних технологій», «Вища математика», «Програмування», «Об’єктно-орієнтоване програмування».

Метою викладання даної навчальної дисципліни є вивчення принципів побудови паралельних алгоритмів, застосування спеціальних засобів синхронізації (семафорів, м’ютексів, моніторів), керування процесами та потоками, використання сучасних програмних бібліотек та мов програмування.

Запропонований матеріал буде корисним як студентам, що вивчають програмування, так і професійним програмістам, які займаються розробкою паралельних комп’ютерних додатків, систем розподіленої обробки інформації.

# САМОСТІЙНА РОБОТА № 1

**Тема:** знайомство з інтерпретатором мови Python.

**Мета:** ознайомитись із синтаксисом мови Python, її основними конструкціями та навчитись запускати на виконання розроблену програму.

## Хід роботи

1) Встановити інтерпретатор мови Python (версія 2.x). Рекомендується встановити два інтерпретатори: стандартний інтерпретатор та IronPython. Обидва інтерпретатори вільно розповсюджуються в мережі Інтернет і доступні на офіційних сайтах: www.python.org та http://ironpython.net. Установка проходить стандартним чином і не потребує висококваліфікованих навичок.

2) Запустити середовище розробки IDLE.

3) Після запрошення “>>>” введіть арифметичний вираз, наприклад, 10+4 і натисніть Enter. Який результат спостерігається? Поекспериментуйте з цією можливістю. Спробуйте обчислити різні математичні виразами із операціями +, -, *, /. Як працює оператор ділення? Який результат він дає? Також можна використати такі операції: ** (піднесення до ступеню), %(остача від ділення). Більш детально з доступними операціями мови Python можна ознайомитись в документації, яка постачається разом із інтерпретатором.

4) Опишіть змінні **x=5** та **y=4** та виконайте операцію ділення. Це можна зробити шляхом введення наступних рядків:

```
>>> x = 5
>>> y = 4
>>> x / y
```

Який результат спостерігається?

5) Повторіть вище наведені рядки, але записавши змінну **x** в форматі дійсного числа (з дробовою частиною):

```
>>> x = 5.0
>>> y = 4
>>> x / y
```

Як при цьому змінився результат?

6) Попереднього результату можна досягти шляхом явного перетворення змінної до дійсного типу float:

```
>>> x = 5
>>> y = 4
>>> float(x) / y
```

Самостійно спробуйте варіанти команд x / float(y) та float (x / y). Які результати дають ці команди? Чи рівноцінні такі команди?

7) Використовуючи документацію, самостійно розглянути інформацію про типи int, long, float. В чому відмінність між типами int та long? Використати функцію type() для того, щоб дізнатися типи об'єктів.

8) Вивести на екран змінні x та y. Це можна зробити шляхом команди:

```
>>> print x
>>> print y
```

Чи змінилися значення змінних після виконання попередніх операцій ділення?

9) Виконати операцію ділення з присвоєнням:

```
>>> x = x / y
```

Вивести на екран змінну **x**. Чи змінилося її значення?

10) Використати список (list) для зберігання послідовності чисел 7; 9; 87; 4 та вивести його на екран:

```
>>> L1 = [7, 9, 87, 4]
>>> print L1
```

11) Вивести на екран тільки число 87:

```
>>> print L1[2]
```

Зверніть увагу, яким чином нумеруються елементи списку. Який номер має початковий елемент 0 чи 1?

12) Вивести на екран кількість елементів в списку:

```
>>> len(L1)
```

13) Вивести всі елементи списку, використовуючи цикл while:

```
>>> i = 0
>>> while (i < len(L1)):
    print L1[i]
    i = i + 1
```

Зверніть увагу, що після того як буде поставлено символ “**:**” і натиснуто Enter, то автоматично буде поставлено відступ для наступного рядка, який показує інтерпретатору, що далі йде тіло циклу. Відступи є аналогом “{” та “}” в мовах C/C++ або begin/end в мові Pascal.

14) Вивести всі елементи списку, використовуючи цикл for:

```
>>> for x in L1:
    print x
```

15) Поекспериментувати з циклами while та for. Ознайомитись з їх конструкціями, використовуючи документацію.

16) Змініть якийсь елемент в списку. Наприклад:

```
>>> L1[1] = 999
```

І знову виведіть весь список на екран.

17) Поекспериментувати зі списками. Чи можуть бути елементи в списку різного типу? Спробуйте поряд із числами використати рядковий тип, значення якого записується в одинарних або подвійних лапках.

18) Створити список, в якому кожен елемент є також списком для зберігання матриці 3x3:

```
>>> L2 = [[1, 2, 3], [4, 5, 6], [7,8,9]]
```

19) Вивести на екран число 6, яке міститься в списку, використовуючи подвійну нумерацію:

```
>>> L2[1][2]
```

20) Вивести всі елементи списку використовуючи вкладені цикл while:

```
>>> i = j = 0
>>> while (i<len(L2)):
 while (j<len(L2[i])):
    print L2[i][j]
    j = j + 1
 i = i + 1
 j = 0
```

Зверніть увагу, як оформлюються відступи при вкладених циклах. Також варто відмітити, що у внутрішньому циклі використовується умова j<len(L2[i]). Пояснити чому це важливо. Підказка: що буде, якщо зберігається не квадратна матриця; чи коректно в цьому випадку буде виконуватись програма.

21) Самостійно написати програму для виведення всіх елементів списку на екран з використанням циклу for.

22) Для виконання додаткових математичних функцій підключити модуль math.

```
>>> import math
```

23) Обчислити значення квадратного кореня із числа 16:

```
>>> math.sqrt(16)
```

24) Використати інструкцію import для приєднання окремих імен із модуля math. При цьому функції можна викликати без назви модуля.

```
>>> from math import sin, cos
>>> sin(1)
>>> cos(1)
```

25) Використовуючи документацію ознайомитись із можливостями модуля math та поекспериментувати з його функціями.

26) Створити власну функцію, яка знаходить суму трьох чисел:

```
>>> def MySum(a,b,c):
  return a+b+c
```

27) Використати створену функцію на конкретних числах та вивести результат її виконання:

```
>>> print MySum(5,3,9)
```

28) Створити функцію, яка обчислює суму чисел від 1 до N:

```
>>> def MySum2(N):
  i = 1
  S = 0
  while (i <= N):
     S = S + i
     i = i + 1
  return S
```

29) Перевірте створену функцію на конкретних числах:

```
>>> print MySum2(3)
>>> print MySum2(5)
>>> print MySum2(10)
```

30) (***Завдання підвищеної складності***) Використовуючи файловий менеджер створити вручну два файли in.txt та out.txt. У файлі in.txt записати три числа, наприклад, 5, 7, 8, кожне число записуючи з нового рядка. У цьому файлі будуть міститися вхідні дані для подальшого кроку. Отримати доступ до файлу in.txt, зчитати всі числа,

збільшити кожне з них на одиницю та записати результати у файл out.txt. Використовуючи документацію, самостійно ознайомитись із додатковими можливостями роботи із файлами. Проекспериментувати з різними можливостями введення у файл та зчитування з нього.

*Примітка:* для обробки файлів необхідно добре розібратися в роботі із рядками символів.

31) Самостійно розібратися з умовним оператором if.

32) Написати програму (сценарій) для подальшого багатократного виконання. Для цього можна використовувати як можливості середовища IDLE, так і будь-який текстовий редактор, наприклад, notepad.exe. При використанні середовища IDLE необхідно клацнути на пункті меню File – New Window або комбінація клавіш Ctrl+N. У вікні, що відкрилось ввести команди, які будуть виконуватись при запуску програми. Нижче наведена програма, яка обчислює суму чисел від 1 до 10.

```
i = 1
S = 0
while (i <= 10):
   S = S + i
   i = i + 1
print S
```

33) Після того як програма написана, необхідно зберегти її з розширенням *.py, наприклад, MyProgram.py. Запуск програми можливий двома способами: із середовища IDLE та з командного рядка.

34) Для запуску написаного сценарію із середовища IDLE необхідно у вікні редагування модулю обрати меню Run – Run Module або клавіша F5. При цьому результат буде виведено в головному вікні інтерпретатора.

35) Іншим способом є запуск із командного рядка. Для цього необхідно запустити cmd.exe. Перейти до каталогу де розміщено інтерпретатор, наприклад, C:\Python27 або C:\IronPython2.7 і виконати команду:

`python ШЛЯХ_ДО_ФАЙЛУ\MyProgram.py` – для стандартного інтерпретатора Python

`ipy ШЛЯХ_ДО_ФАЙЛУ\MyProgram.py` – для інтерпретатора IronPython

36) Індивідуальне завдання. Рівень складності обирається студентом самостійно: рівень А — середній рівень, рівень Б — високий.

36.1) ***Рівень А.*** Написати програму, яка обробляє три списки L1, L2, L3. Значення в кожному списку мають бути встановлені наступним чином: L1 – ASCII-коди прізвища студента; L2 – ASCII-коди імені; L3 – ASCII-коди по-батькові. Вони вводяться вручну при написанні програми за допомогою таблиці 1.1, наведеної нижче. Програма має виводити користувачу наступну інформацію: 1) прізвище, ім'я, по-батькові студента; 2) кількість чисел у списку L1, які кратні трьом; 2) значення, яке найчастіше повторюється у списку L2; 3) список L3, кожне значення якого збільшене на N (де N – номер варіанта)

36.2) ***Рівень Б.*** Виконати завдання рівня А, але значення списків мають бути записані не в програмі, а у файлі in.txt, а результати мають бути виведені в файл out.txt.

Таблиця 1.1 – ASCII-коди символів

| Ctrl | Dec | Hex | Char | Code |
|---|---|---|---|---|
| ^@ | 0 | 00 | | NUL |
| ^A | 1 | 01 | | SOH |
| ^B | 2 | 02 | | STX |
| ^C | 3 | 03 | | ETX |
| ^D | 4 | 04 | | EOT |
| ^E | 5 | 05 | | ENQ |
| ^F | 6 | 06 | | ACK |
| ^G | 7 | 07 | | BEL |
| ^H | 8 | 08 | | BS |
| ^I | 9 | 09 | | HT |
| ^J | 10 | 0A | | LF |
| ^K | 11 | 0B | | VT |
| ^L | 12 | 0C | | FF |
| ^M | 13 | 0D | | CR |
| ^N | 14 | 0E | | SO |
| ^O | 15 | 0F | | SI |
| ^P | 16 | 10 | | DLE |
| ^Q | 17 | 11 | | DC1 |
| ^R | 18 | 12 | | DC2 |
| ^S | 19 | 13 | | DC3 |
| ^T | 20 | 14 | | DC4 |
| ^U | 21 | 15 | | NAK |
| ^V | 22 | 16 | | SYN |
| ^W | 23 | 17 | | ETB |
| ^X | 24 | 18 | | CAN |
| ^Y | 25 | 19 | | EM |
| ^Z | 26 | 1A | | SUB |
| ^[ | 27 | 1B | | ESC |
| ^\ | 28 | 1C | | FS |
| ^] | 29 | 1D | | GS |
| ^^ | 30 | 1E | ▲ | RS |
| ^- | 31 | 1F | ▼ | US |

| Dec | Hex | Char |
|---|---|---|
| 32 | 20 | |
| 33 | 21 | ! |
| 34 | 22 | " |
| 35 | 23 | # |
| 36 | 24 | $ |
| 37 | 25 | % |
| 38 | 26 | & |
| 39 | 27 | ' |
| 40 | 28 | ( |
| 41 | 29 | ) |
| 42 | 2A | * |
| 43 | 2B | + |
| 44 | 2C | , |
| 45 | 2D | - |
| 46 | 2E | . |
| 47 | 2F | / |
| 48 | 30 | 0 |
| 49 | 31 | 1 |
| 50 | 32 | 2 |
| 51 | 33 | 3 |
| 52 | 34 | 4 |
| 53 | 35 | 5 |
| 54 | 36 | 6 |
| 55 | 37 | 7 |
| 56 | 38 | 8 |
| 57 | 39 | 9 |
| 58 | 3A | : |
| 59 | 3B | ; |
| 60 | 3C | < |
| 61 | 3D | = |
| 62 | 3E | > |
| 63 | 3F | ? |

| Dec | Hex | Char |
|---|---|---|
| 64 | 40 | @ |
| 65 | 41 | A |
| 66 | 42 | B |
| 67 | 43 | C |
| 68 | 44 | D |
| 69 | 45 | E |
| 70 | 46 | F |
| 71 | 47 | G |
| 72 | 48 | H |
| 73 | 49 | I |
| 74 | 4A | J |
| 75 | 4B | K |
| 76 | 4C | L |
| 77 | 4D | M |
| 78 | 4E | N |
| 79 | 4F | O |
| 80 | 50 | P |
| 81 | 51 | Q |
| 82 | 52 | R |
| 83 | 53 | S |
| 84 | 54 | T |
| 85 | 55 | U |
| 86 | 56 | V |
| 87 | 57 | W |
| 88 | 58 | X |
| 89 | 59 | Y |
| 90 | 5A | Z |
| 91 | 5B | [ |
| 92 | 5C | \ |
| 93 | 5D | ] |
| 94 | 5E | ^ |
| 95 | 5F | _ |

| Dec | Hex | Char |
|---|---|---|
| 96 | 60 | ` |
| 97 | 61 | a |
| 98 | 62 | b |
| 99 | 63 | c |
| 100 | 64 | d |
| 101 | 65 | e |
| 102 | 66 | f |
| 103 | 67 | g |
| 104 | 68 | h |
| 105 | 69 | i |
| 106 | 6A | j |
| 107 | 6B | k |
| 108 | 6C | l |
| 109 | 6D | m |
| 110 | 6E | n |
| 111 | 6F | o |
| 112 | 70 | p |
| 113 | 71 | q |
| 114 | 72 | r |
| 115 | 73 | s |
| 116 | 74 | t |
| 117 | 75 | u |
| 118 | 76 | v |
| 119 | 77 | w |
| 120 | 78 | x |
| 121 | 79 | y |
| 122 | 7A | z |
| 123 | 7B | { |
| 124 | 7C | \| |
| 125 | 7D | } |
| 126 | 7E | ~ |
| 127 | 7F | ⌂* |

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

## Контрольні питання

1. Які основні типи даних є у мові програмування Python?
2. Які ітераційні конструкції Python Вам відомі?
3. Які функції містить модуль math?
4. Що таке інтерпретатор? У чому відмінність від компілятора?
5. Перерахувати операції над рядковими змінними у Python.

## САМОСТІЙНА РОБОТА № 2

**Тема:** реалізація алгоритмів матричних операцій на мові Python.

**Мета:** ознайомитись із модулем array, програмно реалізувати алгоритми, які базуються на використанні одновимірних та двовимірних масивів.

### Хід роботи

1) Записати в пам'ять два вектор-стовпця V1 та V2 розміром K за варіантом (таблиця 2.1), використовуючи одновимірні масиви модуля array.

Приклад:

```
import array
#K = 3
V1 = array.array('i')
V2 = array.array('f')
V1 = [1,2,3]
V2 = [3.5,1.75,9.01]
```

Самостійно розглянути інформацію про модуль array. Які коди типів може приймати конструктор array, окрім вище зазначених `'i'` та `'f'`? Скільки байтів замає кожен тип в пам'яті? Як впливає використання модуля array на продуктивність виконання програми в порівнянні із звичайними списками?

2) Виконати суму вектор-стовпців, записаних на попередньому кроці. Результат записати у V1 та вивести на екран:

Приклад:

```
i = 0
while (i <= len(V1)-1):
    V1[i] = V1[i] + V2[i]
    i = i + 1
print V1
```

3) Записати в пам'ять дві матриці M1 та M2 розміром NxK за варіантом (таблиця 2.1), використовуючи двовимірні масиви модуля array.

Приклад:

```
import array
# N = 2; K = 3
M1 = array.array('i')
M2 = array.array('f')
M1 = [[2,4,1],[5,7,3]]
M2 = [[0.5,3.2,1.4],[10.599,1.64,3.29]]
```

4) Виконати суму матриць, записаних на попередньому кроці. Результат записати у M1 та вивести на екран:

Приклад:

```
i = 0
j = 0
while(i <= len(M1)-1):
    while(j <= len(M1[0])-1):
        M1[i][j] = M1[i][j] + M2[i][j]
        j = j + 1
    j = 0
    i = i + 1
print M1
```

5) Написати програму, яка виконує множення матриці M1 на вектор-стовпець V2, які було створено на попередньому кроці.

6) Написати програму, яка виконує добуток матриць M2 та M4, де M4 – матриця розміром KxQ, яка задана в таблиці 2.1.

7) Записати у пам'ять одновимірний масив V3 розміром J, кожне із значень якого ініціалізовано виразом F1(i) (i – номер елемента масиву) за варіантом (таблиця 2.2), використовуючи для цього цикл. Вивести на екран три останніх елемента масиву.

Приклад:

```
import array
# F1: x |-> y;       y = 2x
J=100
V3 = array.array('i')
V3.append(0)
i = 1
while(i <= J-1):
    V3.append(2*i)
    i = i + 1
print V3[len(V3)-3]
print V3[len(V3)-2]
print V3[len(V3)-1]
```

8) Виконати попередній крок із використанням ініціалізатора:

Приклад:

```
import array
# F1: x |-> y;       y = 2x
J=100
V3 = array.array('i', (2*i for i in range(J)))
print V3[len(V3)-3]
print V3[len(V3)-2]
print V3[len(V3)-1]
```

Проекспериментувати з ініціалізатором.

9) Записати у пам'ять матрицю M3 розміром N1xN2, кожне із значень якого ініціалізовано виразом F2(i,j) (i, j – номери строки та стовпця елемента масиву) за варіантом (таблиця 2.2). Вивести на екран два крайні елементи головної діагоналі та два крайні елементи побічної діагоналі.

Приклад:

```
import array
# F2: x,y |-> z;       z = x + y
N1 = 100
N2 = 50
```

```
M3 = array.array('i')
M3 = [[i+j for i in range(N2)] for j in range(N1)]
print M3[0][0]
print M3[N1-1][N2-1]
print M3[0][N2-1]
print M3[N1-1][0]
```

10) Використовуючи документацію та додаткову літературу по мові Python, ознайомитись з призначенням модуля time.

11) Розробити функцію Func, яка залежить від параметру k та потребує для свого виконання 5 і більше секунд (чим більший параметр k, тим більший час виконання функції). Заміряти час виконання t в залежності від параметру k та побудувати графік залежності t(k) та звести отримані дані в таблицю.

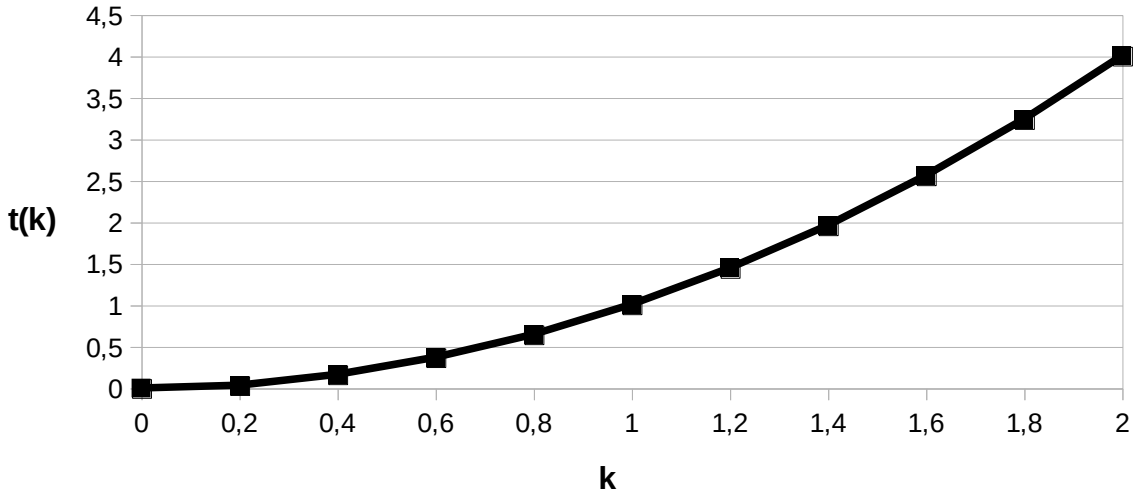
Приклад. Нехай функція Func(k) виконується $k^2$ секунд. Для програмної реалізації використано метод sleep.

```
import time
def Func(k):
    time.sleep(k*k)
i = 0
print "k\tt(k)"
while (i <= 2):
    t1 = time.clock()
    Func(i)
    t2 = time.clock()
    print i, "\t", round((t2-t1),3)
    i = i + 0.2
```

Результат:

```
k t(k)
0   0.0
0.2   0.034
0.4   0.164
0.6   0.37
```

```
0.8    0.646
1.0    1.008
1.2    1.446
1.4    1.96
1.6    2.562
1.8    3.241
2.0    4.005
```




Увага! При побудові власної функції Func метод sleep використовувати не можна.

12) **Завдання підвищеної складності.** Побудувати графік залежності для попереднього кроку, використовуючи можливості мови Python.

*Примітка:* для виконання завдання необхідно самостійно розібратися із модулями для виводу графіки (наприклад, turtle або Tkinter).

Таблиця 2.1 – Індивідуальні завдання

| **Варіант** | **V1** | **V2** | **M1** | | | **M2** | | | **M4** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -19 | 2.367 | 27 | 13 | -21 | 4.511 | 8.550 | 6.380 | 25 | 25 | -16 |
| | 17 | 15.399 | 23 | -11 | -21 | 1.952 | 4.446 | 0.594 | 29 | -19 | -11 |
| | 29 | 9.320 | 17 | 10 | -2 | 1.526 | 4.179 | 7.408 | 26 | -17 | 6 |
| 2 | 11 | 3.593 | 24 | -27 | -12 | 9.503 | 8.503 | 7.230 | -30 | 13 | -29 |
| | 0 | 17.734 | 22 | -3 | 20 | 0.471 | 2.532 | 2.607 | -25 | -9 | -1 |
| | 8 | 5.289 | 19 | 23 | 15 | 3.047 | 3.794 | 6.492 | 20 | -21 | -1 |

| Варіант | V1 | V2 | M1 | | | M2 | | | M4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 6 | 1.892 | 4 | 24 | 21 | 0.046 | 7.341 | 8.810 | 0 | 16 | 3 |
| | 26 | 10.984 | 0 | -17 | -13 | 7.868 | 7.332 | 1.350 | -28 | -24 | 3 |
| | -7 | 19.731 | 26 | -13 | -3 | 8.744 | 9.941 | 9.419 | 19 | 27 | 9 |
| 4 | -3 | 0.484 | -11 | -17 | -2 | 6.612 | 1.040 | 9.134 | -19 | 0 | 23 |
| | 0 | 10.085 | -14 | -26 | 14 | 9.828 | 4.157 | 5.760 | 9 | 0 | 1 |
| | -6 | 12.110 | -21 | -29 | 30 | 8.897 | 3.793 | 9.994 | -1 | 28 | 5 |
| 5 | 16 | 18.840 | 5 | -6 | -3 | 8.361 | 5.666 | 8.227 | 2 | 20 | -17 |
| | 22 | 19.043 | -26 | 16 | -30 | 2.813 | 2.419 | 3.371 | 4 | -10 | 13 |
| | 6 | 10.271 | 18 | -14 | -15 | 2.379 | 0.442 | 8.794 | -30 | -1 | -27 |
| 6 | 10 | 12.177 | -9 | 30 | -20 | 0.422 | 7.423 | 2.414 | -30 | 20 | 13 |
| | 29 | 13.830 | 17 | -25 | -26 | 8.435 | 9.432 | 2.009 | -2 | 1 | 9 |
| | 1 | 18.520 | 11 | -4 | -10 | 9.388 | 7.157 | 6.627 | 1 | -6 | -5 |
| 7 | -20 | 12.986 | 27 | 26 | -7 | 3.174 | 0.193 | 6.175 | -8 | -24 | -3 |
| | -28 | 9.879 | -27 | 3 | -22 | 3.532 | 4.503 | 4.147 | -19 | 19 | 0 |
| | -26 | 14.782 | 9 | -24 | -14 | 7.286 | 0.525 | 4.215 | 2 | 10 | 7 |
| 8 | -26 | 14.427 | 3 | 19 | 4 | 8.212 | 4.828 | 7.938 | -22 | 17 | -9 |
| | -18 | 12.998 | 9 | -21 | 0 | 7.721 | 7.368 | 9.674 | 29 | -9 | 17 |
| | -29 | 10.415 | -14 | -9 | -12 | 4.872 | 6.390 | 6.457 | 28 | 17 | -23 |
| 9 | 8 | 18.758 | 11 | 26 | -3 | 3.120 | 7.825 | 8.705 | 13 | -1 | 19 |
| | -2 | 9.604 | 24 | 0 | -1 | 4.455 | 0.035 | 6.756 | -6 | -24 | -28 |
| | -29 | 6.683 | 20 | 11 | 4 | 1.338 | 7.195 | 5.179 | 12 | 2 | -14 |
| 10 | -16 | 10.849 | 25 | -22 | 28 | 4.298 | 9.410 | 5.432 | -15 | 24 | -21 |
| | -2 | 4.396 | -17 | -12 | -29 | 1.393 | 2.211 | 9.442 | 8 | 0 | -14 |
| | 28 | 0.516 | -28 | 24 | 11 | 2.314 | 7.468 | 3.050 | 21 | 4 | 7 |
| 11 | -8 | 7.340 | -19 | -25 | 20 | 7.495 | 4.487 | 6.311 | 5 | -17 | -26 |
| | 25 | 1.450 | 11 | 19 | 14 | 9.582 | 6.854 | 7.591 | 25 | -17 | 12 |
| | 21 | 19.852 | -24 | 7 | -5 | 5.801 | 0.913 | 0.039 | 16 | -13 | -15 |
| 12 | 24 | 5.422 | 12 | 7 | -11 | 9.914 | 7.624 | 6.530 | 27 | 19 | -25 |
| | -12 | 4.277 | 29 | -23 | -29 | 2.099 | 2.989 | 3.333 | 11 | 20 | -20 |
| | 0 | 19.775 | -25 | -22 | 5 | 6.126 | 7.112 | 2.997 | -22 | 1 | -30 |
| 13 | 18 | 13.787 | -2 | 23 | -28 | 0.548 | 0.156 | 0.852 | -9 | -3 | 15 |
| | 11 | 4.777 | 30 | 30 | 24 | 5.353 | 6.465 | 1.745 | -11 | 5 | 12 |
| | -30 | 14.543 | -23 | 19 | -25 | 7.318 | 8.699 | 5.438 | -29 | 28 | 8 |
| 14 | -14 | 6.638 | 14 | 1 | -20 | 7.857 | 8.953 | 5.442 | -8 | 2 | 19 |
| | -26 | 19.180 | -24 | -17 | 13 | 1.651 | 2.927 | 6.594 | -12 | -4 | -15 |

| Варіант | V1 | V2 | M1 | | | M2 | | | M4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | -24 | 10.493 | 3 | 14 | -19 | 8.716 | 3.779 | 0.127 | 1 | 23 | 4 |
| 15 | -17 | 1.628 | -30 | 25 | 2 | 7.748 | 7.052 | 4.985 | -27 | -19 | 7 |
| | -30 | 17.227 | 2 | -3 | 11 | 1.460 | 3.536 | 9.793 | -9 | -12 | 28 |
| | 16 | 17.181 | -24 | 18 | 21 | 1.142 | 0.735 | 6.686 | -11 | 17 | 27 |
| 16 | -16 | 0.202 | 4 | 15 | 25 | 1.327 | 4.882 | 1.448 | -11 | -19 | -26 |
| | -19 | 16.322 | 16 | 2 | 30 | 5.255 | 0.703 | 8.516 | 16 | -29 | 11 |
| | 7 | 17.047 | 27 | -24 | 12 | 5.630 | 1.247 | 6.974 | 10 | 20 | 17 |
| 17 | 30 | 19.858 | 23 | -27 | -16 | 1.551 | 9.535 | 5.519 | -1 | -6 | -4 |
| | -25 | 9.628 | 12 | 27 | 18 | 7.407 | 1.857 | 0.705 | 22 | 25 | 14 |
| | -6 | 11.054 | -30 | -10 | 8 | 7.486 | 4.796 | 3.200 | 21 | -27 | 9 |
| 18 | -15 | 4.440 | 25 | -10 | -7 | 6.540 | 4.750 | 7.784 | -5 | -11 | -12 |
| | 3 | 14.291 | 14 | -30 | -21 | 7.015 | 4.100 | 7.456 | 1 | 26 | 22 |
| | 19 | 9.126 | 0 | 13 | 26 | 4.293 | 0.406 | 6.926 | -13 | 0 | -29 |
| 19 | -11 | 10.702 | -19 | -22 | -11 | 8.674 | 0.766 | 3.707 | -26 | -24 | 2 |
| | 15 | 10.669 | -22 | -30 | -13 | 1.187 | 5.457 | 6.653 | 11 | 26 | 2 |
| | -7 | 4.422 | -11 | -29 | -15 | 9.731 | 6.414 | 2.114 | -16 | 13 | 15 |
| 20 | -5 | 11.802 | -13 | -25 | 24 | 9.831 | 6.801 | 8.239 | 27 | 7 | -30 |
| | 19 | 9.760 | -29 | -16 | -3 | 8.986 | 9.368 | 4.988 | 6 | -26 | -8 |
| | 1 | 9.017 | -3 | 21 | -20 | 8.259 | 4.707 | 0.535 | -17 | -3 | 12 |
| 21 | -19 | 16.683 | 3 | -7 | -17 | 6.294 | 1.037 | 4.444 | 24 | -3 | 9 |
| | 17 | 4.988 | -28 | -19 | -24 | 1.563 | 4.668 | 1.673 | -22 | 29 | 24 |
| | 0 | 14.894 | 21 | 20 | -21 | 9.934 | 6.427 | 7.143 | -28 | -20 | 10 |
| 22 | 17 | 0.527 | -4 | 6 | 1 | 3.719 | 5.922 | 6.779 | 26 | 16 | -2 |
| | -8 | 8.185 | -28 | 30 | 15 | 2.422 | 1.317 | 2.672 | -1 | -18 | -23 |
| | -28 | 16.082 | -24 | 22 | -8 | 2.927 | 1.974 | 6.671 | 16 | -20 | -24 |
| 23 | -9 | 13.634 | 18 | 5 | -22 | 6.124 | 3.028 | 5.724 | 19 | 13 | 0 |
| | 13 | 12.036 | 10 | -12 | 25 | 8.391 | 7.507 | 0.178 | 4 | -12 | -14 |
| | -26 | 15.788 | 27 | -9 | 12 | 3.606 | 1.648 | 8.020 | -19 | 9 | -12 |
| 24 | -17 | 11.195 | 13 | -12 | 21 | 5.110 | 5.672 | 9.908 | -30 | -6 | 11 |
| | 4 | 17.651 | -12 | -15 | -4 | 4.562 | 9.334 | 1.018 | -25 | -11 | -26 |
| | -14 | 14.546 | 10 | -26 | -14 | 2.177 | 5.699 | 5.481 | -11 | 10 | 24 |
| 25 | -5 | 15.825 | 5 | -22 | -19 | 3.159 | 8.696 | 6.536 | 3 | 17 | 25 |
| | 22 | 4.652 | -19 | -21 | -14 | 6.516 | 6.201 | 3.821 | 4 | -16 | -1 |
| | 30 | 19.636 | -14 | 23 | -30 | 3.413 | 5.687 | 4.475 | -26 | 7 | -16 |

Таблиця 2.2 – Індивідуальні завдання

| **Варіант** | **J** | **F1(i)** | **N1** | **N2** | **F2(i, j)** |
|---|---|---|---|---|---|
| 1 | 73 | 3+i | 142 | 50 | 2i+j |
| 2 | 95 | 5i | 115 | 43 | i+3j |
| 3 | 96 | 9-i | 102 | 38 | (i-j)*5 |
| 4 | 67 | 3i | 136 | 42 | 3i+j |
| 5 | 114 | 7+i | 105 | 39 | i+3j |
| 6 | 84 | 1-i | 127 | 59 | (i-j)*6 |
| 7 | 109 | 4+i | 144 | 35 | 3i+j |
| 8 | 93 | 3*i | 101 | 50 | i+3j |
| 9 | 96 | 2+i | 142 | 42 | (i-j)*6 |
| 10 | 77 | 77-i | 135 | 58 | 4i+j |
| 11 | 107 | 20+i | 123 | 34 | i+3j |
| 12 | 116 | 6i-20 | 137 | 32 | (i-j)*7 |
| 13 | 91 | 3i+8 | 130 | 49 | 4i+j |
| 14 | 69 | 9i/15 | 132 | 44 | i+3j |
| 15 | 100 | 4i-150 | 107 | 30 | (i-j)*7 |
| 16 | 117 | 6i+32 | 108 | 44 | 5i+j |
| 17 | 88 | 15i/40 | 123 | 44 | i+3j |
| 18 | 107 | 17-4i | 126 | 43 | (i-j)*8 |
| 19 | 70 | 20+i/2 | 147 | 38 | 5i+j |
| 20 | 71 | 45-15i | 129 | 59 | i+3j |
| 21 | 114 | 70i-10 | 123 | 43 | (i-j)*8 |
| 22 | 92 | 10i | 106 | 44 | 6i+j |
| 23 | 87 | 33-i/45 | 147 | 57 | i+3j |
| 24 | 104 | 20i-i/4 | 101 | 34 | (i-j)*9 |
| 25 | 80 | 14i+1 | 148 | 53 | 6i+j |

## Контрольні питання

1. Які етапи створення паралельного алгоритму?
2. Сформулювати закон Амдала.
3. Що таке коефіцієнт прискорення та коефіцієнт ефективності?
4. У чому полягає принцип необмеженого паралелізму?

## САМОСТІЙНА РОБОТА № 3

**Тема:** багатопоточне програмування на мові Python. Синхронізація потоків з використанням семафорів.

**Мета:** ознайомитись із модулем threading; навчитись розробляти паралельні програми, які вимагають синхронізації потоків.

### Хід роботи

1) Написати програму, в якій два потоки T0 і T1 виводять на екран символи * та #, потік T0 виводить десять символів “*”, а T1 – десять символів “#”. Для створення потоків використати модуль threading.

Приклад:

```
import threading

def FuncT0():
    for i in range(10):
        print "*"

def FuncT1():
    for i in range(10):
        print "#"

T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T0.start()
T1.start()
T0.join()
T1.join()
```

Запустити програму на виконання декілька разів. Розглянути структуру паралельної програми. Відповісти на питання: для чого потрібні функції `start()` і `join()`? Яку роль відіграє параметр `target`? Чи можна дати інші ім'я функцій замість `FuncT0` і `FuncT1`?

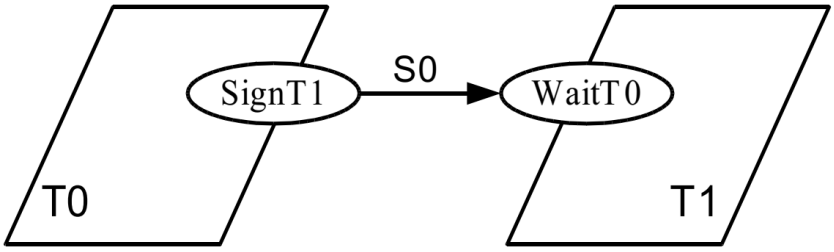
2) В попередню програму додати функції затримки sleep в кожен потік. Встановлюючи різні значення затримки в потоках проаналізувати як змінюється виконання програми.

Приклад:

```
import threading
import time

def FuncT0():
    for i in range(10):
        time.sleep(0.5)
        print "*"

def FuncT1():
    for i in range(10):
        time.sleep(0.9)
        print "#"

T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T0.start()
T1.start()
T0.join()
T0.join()
```

3) Програмно реалізувати схему, коли один потік, чекає іншого як показано в таблиці:

| **T0** | **T1** |
|---|---|
| 1) Затримка на 2 секунди<br>2) Сигнал в T1 (SignT1). | 1) Очікувати T0 (WaitT0).<br>2) Вивести на екран “OK”. |

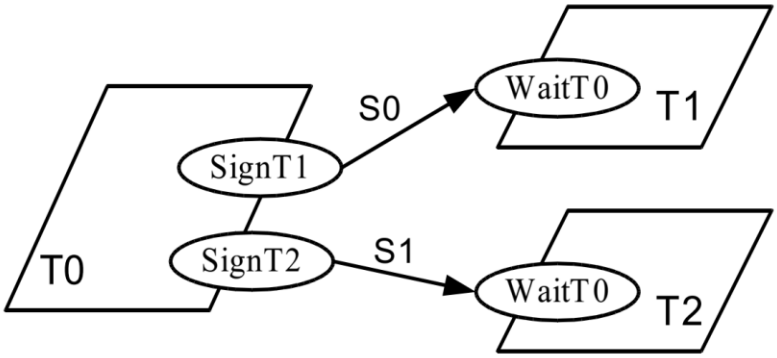



Приклад програмної реалізації:

```
import threading
import time
S0 = threading.Semaphore(0)
def FuncT0():
    time.sleep(2)
    S0.release()
def FuncT1():
    S0.acquire()
    print "OK"
T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T0.start()
T1.start()
T0.join()
T1.join()
```

4) Програмно реалізувати схему, коли два потоки, чекають одного як показано в таблиці:

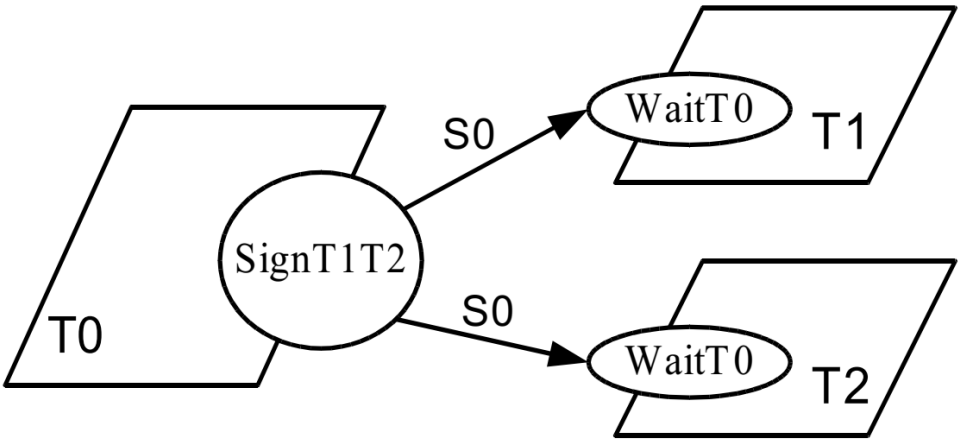
| **T0** | **T1** | **T2** |
|---|---|---|
| 1) Затримка на 2 секунди<br>2) Сигнал в T1 (SignT1).<br>3) Сигнал в T2 (SignT2). | 1) Очікувати T0 (WaitT0).<br>2) Вивести на екран «OK1». | 1) Очікувати T0 (WaitT0).<br>2) Вивести на екран «OK2». |




Приклад програмної реалізації:

```
import threading
import time
S0 = threading.Semaphore(0)
S1 = threading.Semaphore(0)
def FuncT0():
    time.sleep(2)
    S0.release()
    S1.release()
def FuncT1():
    S0.acquire()
    print "OK1"
def FuncT2():
    S1.acquire()
    print "OK2"
T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T2 = threading.Thread(target = FuncT2)
T0.start()
T1.start()
T2.start()
T0.join()
T1.join()
T2.join()
```

5) Попередній крок реалізувати з використанням множинних семафорів.

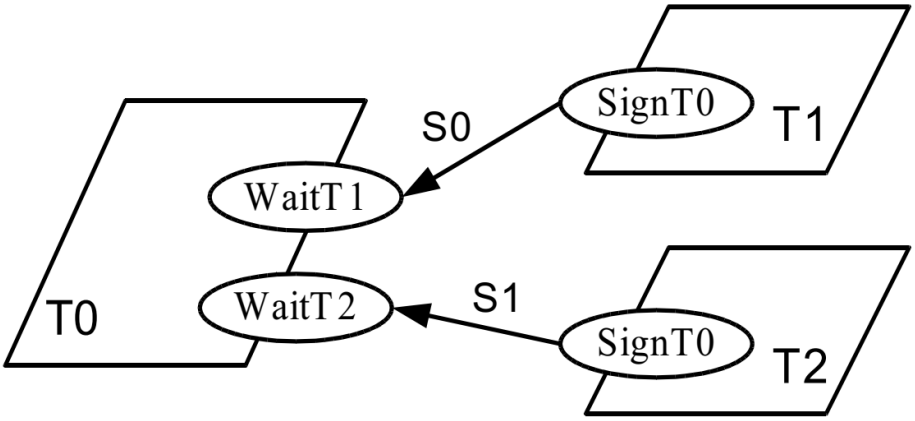



Приклад програмної реалізації:

```
import threading
import time
S0 = threading.Semaphore(0)
def FuncT0():
    time.sleep(2)
    S0.release()
    S0.release()
def FuncT1():
    S0.acquire()
    print "OK1"
def FuncT2():
    S0.acquire()
    print "OK2"
T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T2 = threading.Thread(target = FuncT2)
T0.start()
T1.start()
T2.start()
T0.join()
T1.join()
T2.join()
```

6) Програмно реалізувати схему, коли один потік чекає двох потоків як показано в таблиці:

| T0 | T1 | T2 |
|---|---|---|
| 1) Очікувати T1 (WaitT1).<br>2) Очікувати T2 (WaitT2).<br>3) Вивести на екран “OK”. | 1) Затримка на 2 секунди.<br>2) Сигнал в T0 (SignT0). | 1) Затримка на 2 секунди<br>2) Сигнал в T0 (SignT0). |

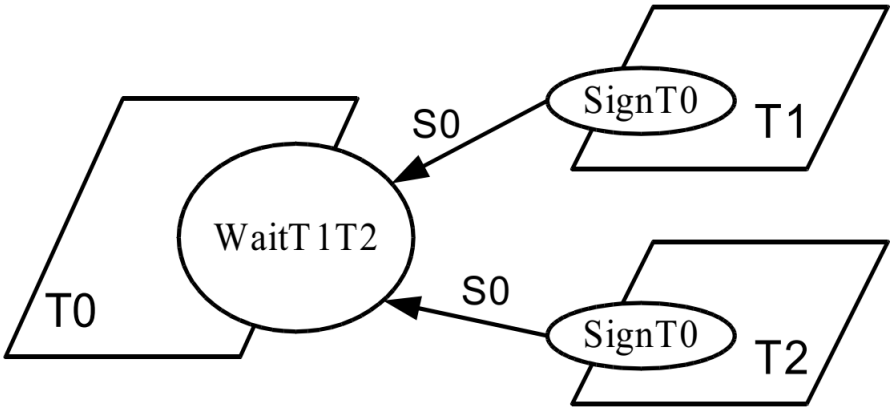



Приклад програмної реалізації:

```
import threading
import time
S0 = threading.Semaphore(0)
S1 = threading.Semaphore(0)
def FuncT0():
    S0.acquire()
    S1.acquire()
    print "OK"
def FuncT1():
    time.sleep(2)
    S0.release()

def FuncT2():
    time.sleep(2)
    S1.release()
T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T2 = threading.Thread(target = FuncT2)
```

```
T0.start()
T1.start()
T2.start()
T0.join()
T1.join()
T2.join()
```

7) Попередній крок реалізувати з використанням множинних семафорів.




Приклад:

```
import threading
import time
S0 = threading.Semaphore(0)
def FuncT0():
    S0.acquire()
    S0.acquire()
    print "OK"
def FuncT1():
    time.sleep(2)
    S0.release()

def FuncT2():
    time.sleep(2)
    S0.release()
T0 = threading.Thread(target = FuncT0)
T1 = threading.Thread(target = FuncT1)
T2 = threading.Thread(target = FuncT2)
```

```
T0.start()
T1.start()
T2.start()
T0.join()
T1.join()
T2.join()
```

8) Задано граф роботи потоків (варіанти завдань для кожного студента представлені в таблиці 3.1). Кожна задача в графі виконується в окремому потоці. Використовуючи семафори виконати синхронізацію потоків. Змінні x, y та z є глобальними для всіх потоків та проініціалізовані наступним чином: x = 12, y = 5, z = 4.

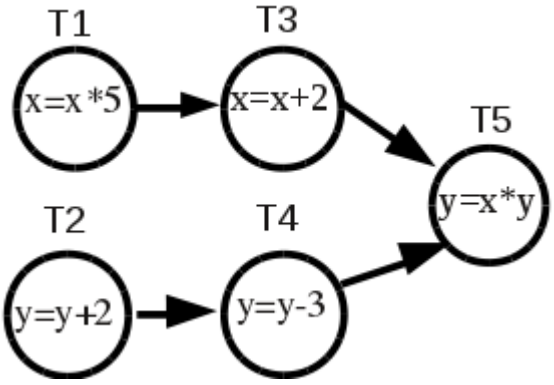
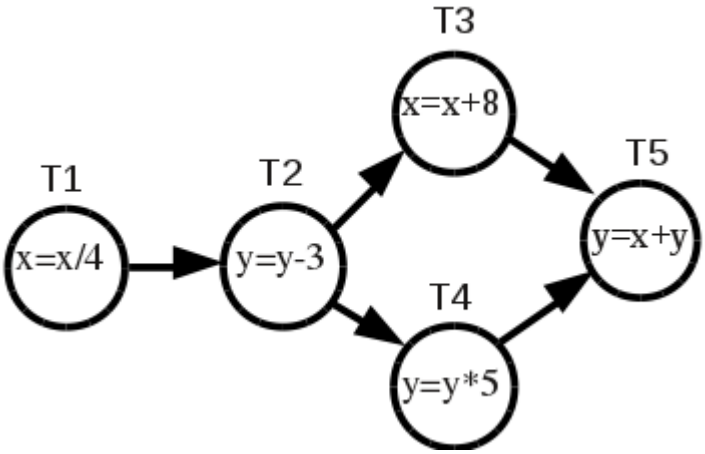
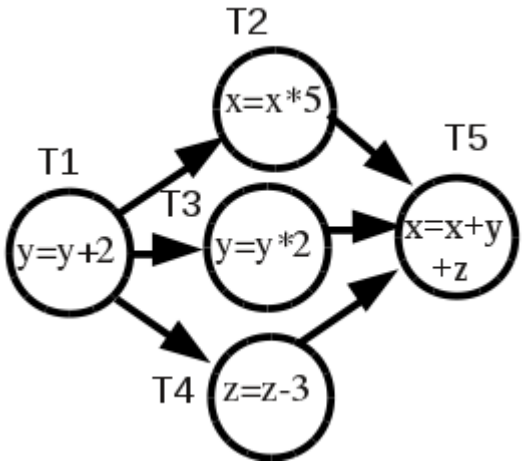
*Примітка:* для вирішення даної задачі необхідно ознайомитися із прикладом, представленим в лекції.

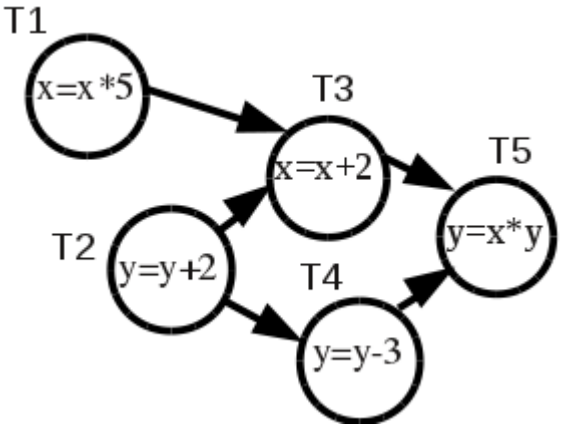
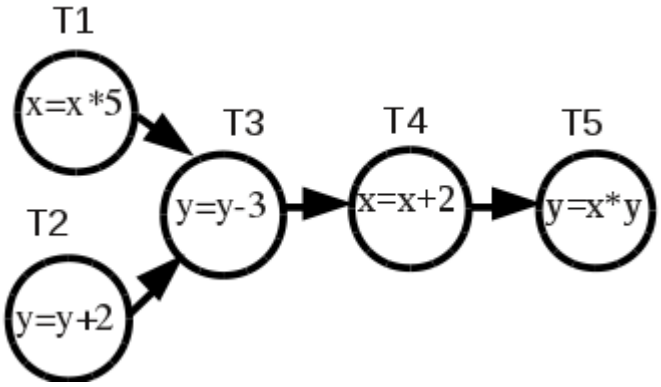
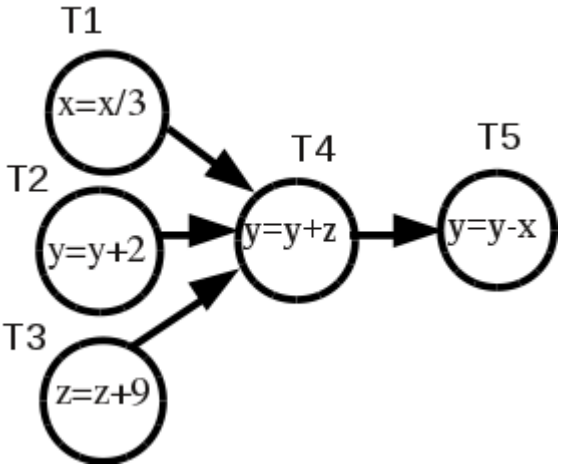
9) Написати паралельну програму, яка двома потоками обчислює векторну операцію (варіанти завдань для кожного студента представлені в таблиці 3.2). Де X, Y, Q — вектори розміром N=20. Причому вектор X ініціалізується одиницями, а Y — числами від 1 до N. Кількість потоків рівна двом. Ініціалізацію векторів виконати паралельно в різних потоках.

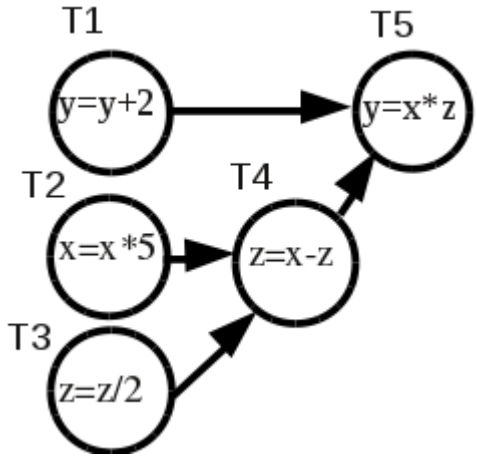
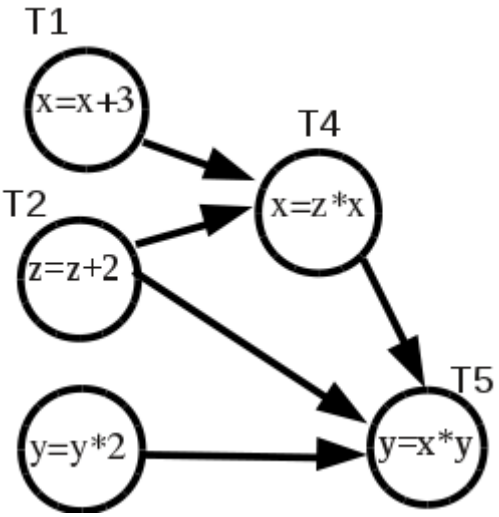
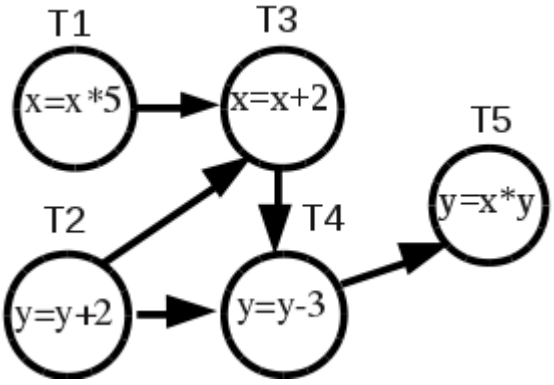
*Примітка:* для вирішення даної задачі необхідно ознайомитися із прикладом, представленим в лекції.

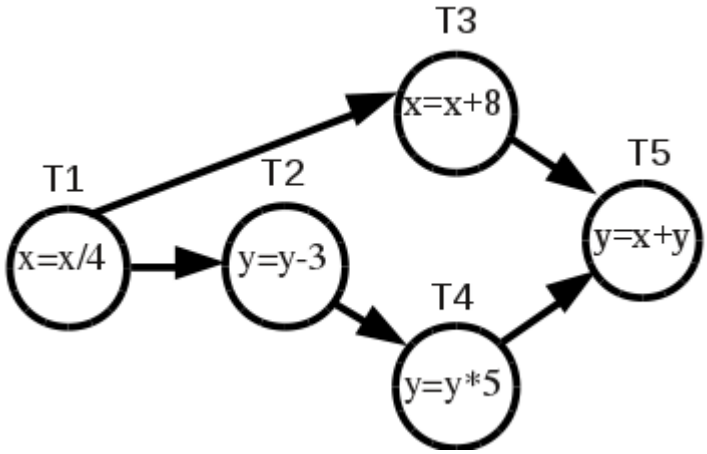
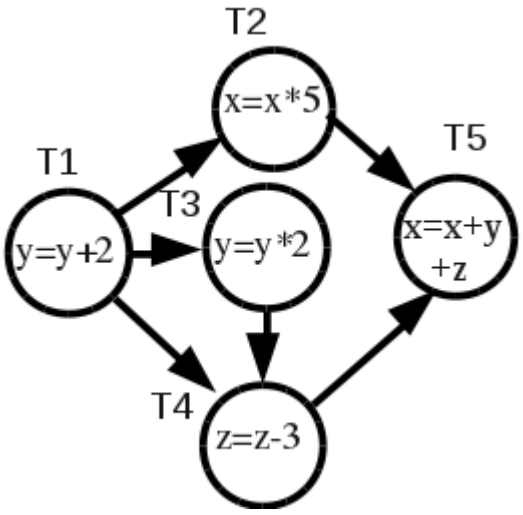
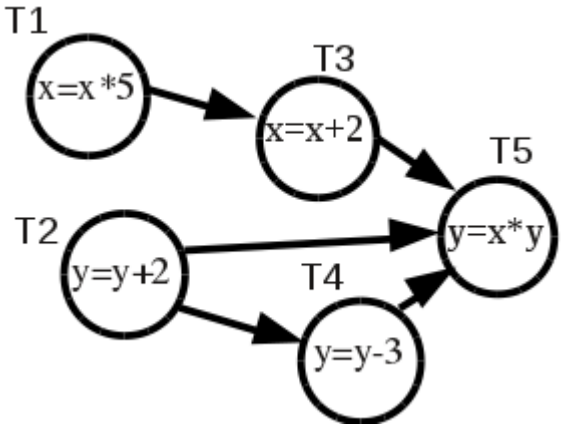
10) **Завдання підвищеної складності.** Попереднє завдання реалізувати для кількості потоків рівній трьом: на початку програми двома потоками виконується ініціалізація, а потім трьома потоками виконуються обчислення.

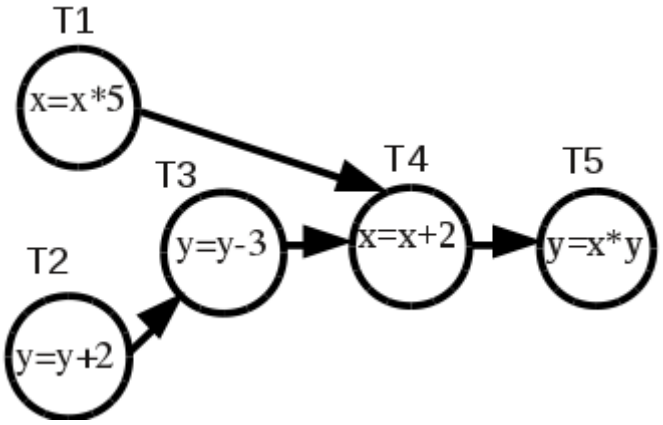
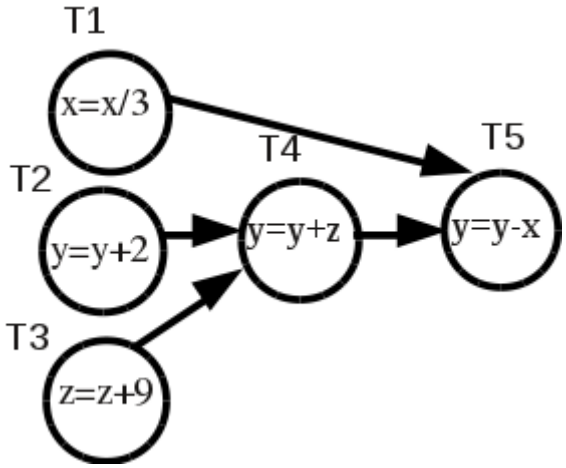
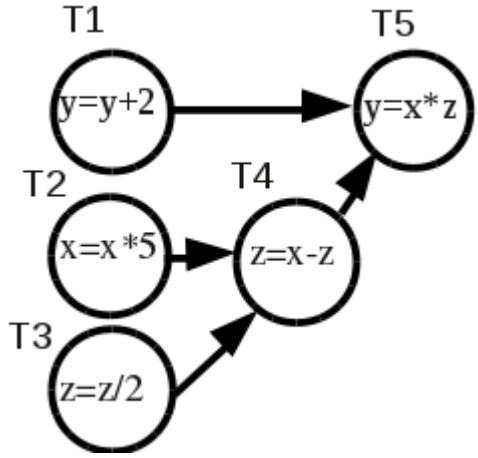
Таблиця 3.1 – Індивідуальні завдання

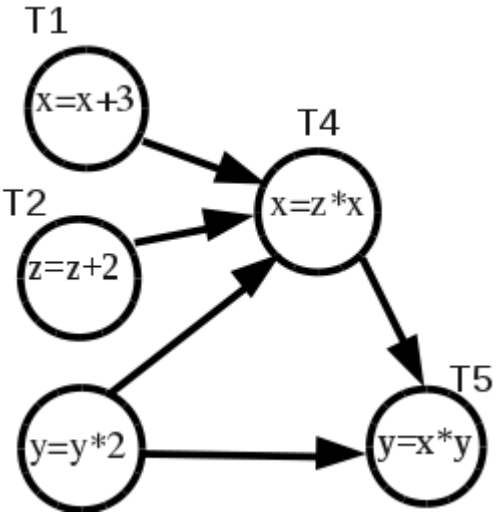
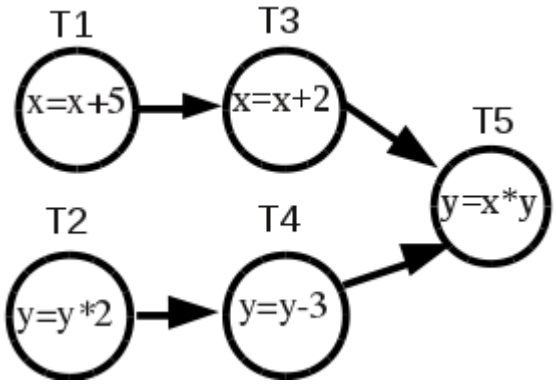
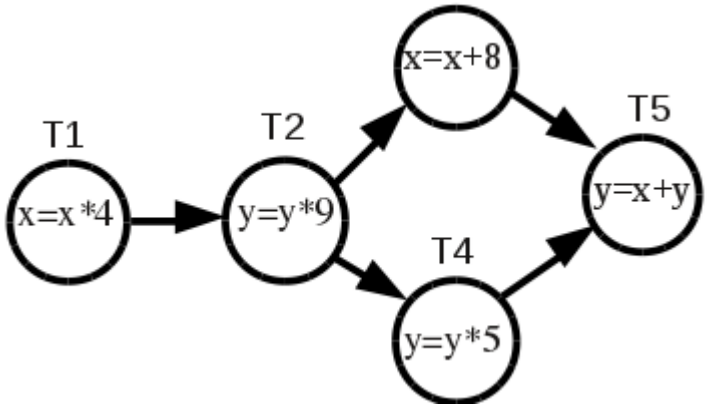
| **Варіант** | **Граф роботи потоків** |
|---|---|
| 1 |  |
| 2 |  |
| 3 |  |

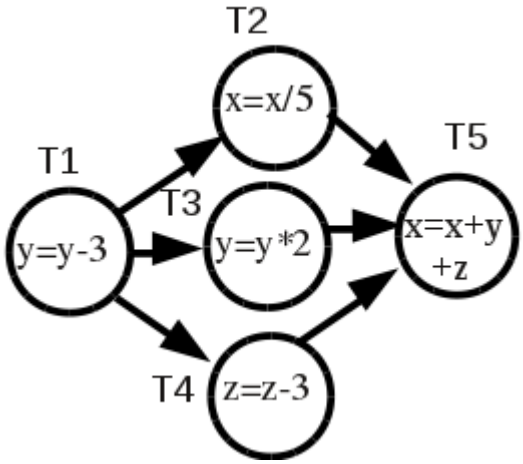
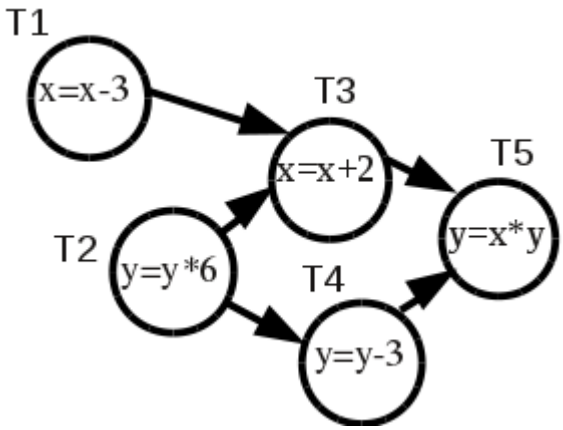
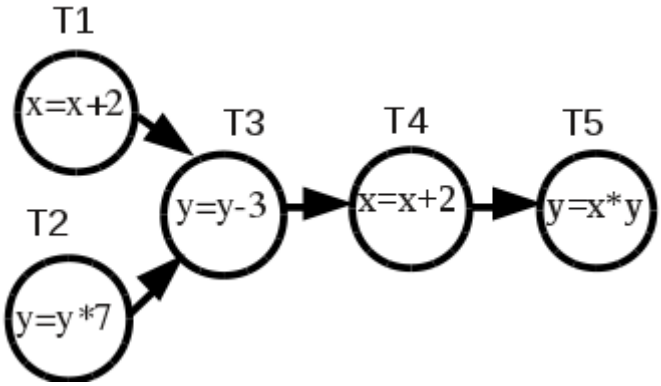
| Варіант | Граф роботи потоків |
|---|---|
| 4 |  |
| 5 |  |
| 6 |  |

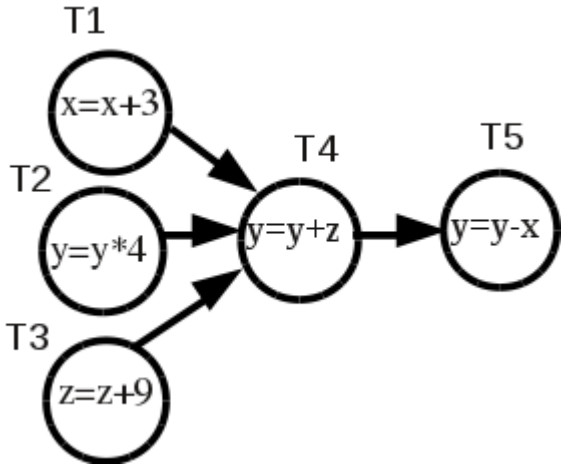
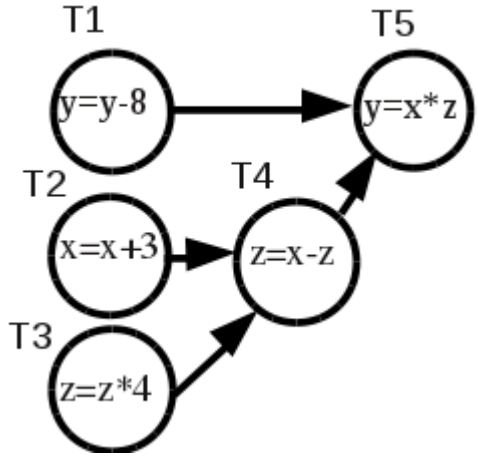
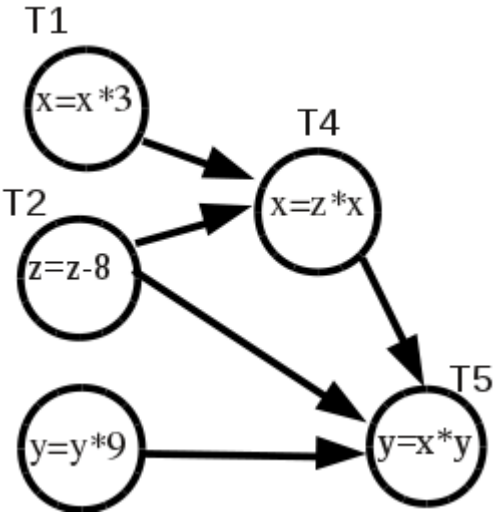
| Варіант | Граф роботи потоків |
|---|---|
| 7 |  |
| 8 |  |
| 9 |  |

| Варіант | Граф роботи потоків |
|---|---|
| 10 |  |
| 11 |  |
| 12 |  |

| Варіант | Граф роботи потоків |
|---|---|
| 13 |  |
| 14 |  |
| 15 |  |

| Варіант | Граф роботи потоків |
|---|---|
| 16 |  |
| 17 |  |
| 18 |  |

| Варіант | Граф роботи потоків |
|---|---|
| 19 |  |
| 20 |  |
| 21 |  |

| Варіант | Граф роботи потоків |
|---|---|
| 22 |  |
| 23 |  |
| 24 |  |

Таблиця 3.2 – Індивідуальні завдання

| **Варіант** | **Операція** | **Варіант** | **Операція** |
|---|---|---|---|
| 1 | *Q=3X+5Y* | 14 | *Q=6X+11Y* |
| 2 | *Q=7X+2Y* | 15 | *Q=2X+5Y* |
| 3 | *Q=X+2Y* | 16 | *Q=11X+11Y* |
| 4 | *Q=2X+3Y* | 17 | *Q=14X+5Y* |
| 5 | *Q=5X+2Y* | 18 | *Q=19X+2Y* |
| 6 | *Q=X+9Y* | 19 | *Q=9X+11Y* |
| 7 | *Q=10X+7Y* | 20 | *Q=17X+3Y* |
| 8 | *Q=12X+Y* | 21 | *Q=8X+5Y* |
| 9 | *Q=2X+8Y* | 22 | *Q=7X+9Y* |
| 10 | *Q=9X+21Y* | 23 | *Q=8X+12Y* |
| 11 | *Q=3X+17Y* | 24 | *Q=7X+16Y* |
| 12 | *Q=18X+7Y* | 25 | *Q=12X+17Y* |
| 13 | *Q=15X+Y* | | |

## Контрольні питання

1. Що таке семафор?
2. Чим відрізняються м’ютекси від семафорів?
3. Що таке синхронізація потоків?
4. У чому сутність «deadlock» та як його уникнути?
5. Які операції вважаються атомарними?

## САМОСТІЙНА РОБОТА № 4

**Тема:** реалізація механізму передачі повідомлень на мові Python. Черги.

**Мета:** розглянути механізм черг в мові Python та реалізувати з його використанням обмін даними між процесами.

### Хід роботи

1) Створити додатковий процес P0 та передати з нього в головний процес номер власного варіанту n.

Приклад (для n = 100):

```
from multiprocessing import Process, Queue

def funcP0(q):
    n=100
    q.put(n)

if (__name__=='__main__'):
    q = Queue()
    P0 = Process(target = funcP0, args = (q,))
    P0.start()
    n = q.get()
    print n
    P0.join()
```

2) Створити додатковий процес P0 та передати з нього в головний процес власне прізвище.

Приклад:

```
from multiprocessing import Process, Queue

def funcP0(q):
    q.put("Shchukin")

if (__name__=='__main__'):
    q = Queue()
```

```
    P0 = Process(target = funcP0, args = (q,))
    P0.start()

    print q.get()
    P0.join()
```

3) Створити додатковий процес P0 та передати з нього в головний процес список, який містить власне прізвище та номер варіанту.

Приклад:

```
from multiprocessing import Process, Queue

def funcP0(q):
    L = [100, "Shchukin"]
    q.put(L)

if (__name__=='__main__'):
    q = Queue()
    P0 = Process(target = funcP0, args = (q,))
    P0.start()

    L = q.get()
    print "Variant: ",L[0]
    print "Prizvishche: ", L[1]
    P0.join()
```

4) Створити додатковий процес P0 та передати з нього в головний процес масив X із десяти чисел (кожен елемент масиву дорівнює номер варіанту).

Приклад:

```
from multiprocessing import Process, Queue
from array import array

def funcP0(q):
    X = array("i",[100 for i in range(10)])
    q.put(X)
```

```
if (__name__=='__main__'):
    q = Queue()
    P0 = Process(target = funcP0, args = (q,))
    P0.start()

    X = q.get()
    print X
    P0.join()
```

5) Написати розподілену програму, яка обчислює значення виразу F за варіантом (таблиця 4.1). Кожна арифметична операція має бути виконана в окремому процесі. Змінні ініціалізуються в тому ж процесі, в якому вони вперше використовуються. Ініціалізувати змінні наступними значеннями: x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5, x6 = 6.

6) Написати розподілену програму, яка обчислює результат векторної операції за варіантом (таблиця 4.1). Кількість елементів в кожному векторі рівна n = 10. Кількість процесів в програмі рівна двом. Елементи векторів X і Y ініціалізувати одиницями в різних процесах.

Таблиця 4.1 – Індивідуальні завдання

| **Варіант** | **Вираз F** | **Векторна операція** |
|---|---|---|
| 1 | x1+x2+x3+x4+x5+x6 | Q=3X+5Y |
| 2 | x1+x2*x3+x4+x5+x6 | Q=7X+2Y |
| 3 | x1*x2+x3*x4+x5+x6 | Q=X+2Y |
| 4 | x1+x2+x3+(x4+x5)*x6 | Q=2X+3Y |
| 5 | (x1+x2)*(x3+x4)+x5*x6 | Q=5X+2Y |
| 6 | (x1+x2+x3)*(x4+x5+x6) | Q=X+9Y |
| 7 | (x1+x2)*(x3+x4)+x5+x6 | Q=10X+7Y |
| 8 | (x1+x2)*(x3+x4)*x5+x6 | Q=12X+Y |
| 9 | (x1+x2)*(x3+x4)*(x5+x6) | Q=2X+8Y |
| 10 | (x1+x2)*(x3+x4+x5+x6) | Q=9X+21Y |
| 11 | (x1*x2+x3)*x4+x5+x6 | Q=3X+17Y |

| Варіант | Вираз F | Векторна операція |
|---|---|---|
| 12 | (x1+x2)+x3*(x4+x5+x6) | Q=18X+7Y |
| 13 | x1*x2+(x3+x4+x5)*x6 | Q=15X+Y |
| 14 | (x1+x2*x3+x4+x5)*x6 | Q=6X+11Y |
| 15 | x1*(x2+x3+x4+x5)*x6 | Q=2X+5Y |
| 16 | (x1*x2+x3)*(x4+x5+x6) | Q=11X+11Y |
| 17 | x1+((x2+x3)*x4+x5)*x6 | Q=14X+5Y |
| 18 | x1*x2*x3+x4*x5*x6 | Q=19X+2Y |
| 19 | x1*x2+(x3+x4+x5)*x6 | Q=9X+11Y |
| 20 | (x1*x2+x3)*(x4+x5)*x6 | Q=17X+3Y |
| 21 | (x1+x2)*x3*x4+x5*x6 | Q=8X+5Y |
| 22 | (x1+x2+x3)*x4*(x5+x6) | Q=7X+9Y |
| 23 | (x1*x2+(x3+x4)+x5)*x6 | Q=8X+12Y |
| 24 | x1+(x2+x3)+(x4+x5)*x6 | Q=7X+16Y |
| 25 | (x1+x2)+(x3*x4+x5*x6) | Q=12X+17Y |

## Контрольні питання

1. Яким чином події застосовуються для синхронізації потоків?
2. Які технології розподіленої обробки даних Вам відомі?
3. Як розпаралелюються матричні операції?
4. Якими засобами мови Python визначити час виконання функції?
5. Пояснити модель паралельних обчислень «виробник-споживач».

## САМОСТІЙНА РОБОТА № 5

**Тема:** реалізації моделі «клієнт-сервер» на мові Python. Канали.

**Мета:** ознайомитись з механізмом каналів як способом взаємодії процесів; реалізувати схеми взаємодіїї клієнтських і серверних процесів за допомогою каналів.

### Хід роботи

1) Створити додатковий процес P1 та передати з нього в головний процес номер власного варіанту n. Для передачі використовувати механізм каналів. Приклад (для n = 100):

```
from multiprocessing import Process, Pipe
def funcP1(pipe):
    n = 100
    pipe[1].send(n)
if (__name__ == '__main__'):
    pipe = Pipe()
    P1 = Process(target = funcP1, args = (pipe,))
    P1.start()
    n = pipe[0].recv()
    P1.join()
    print "n =",n
```

2) Створити додатковий процес P1 та передати з нього в головний процес власне прізвище. Приклад:

```
from multiprocessing import Process, Pipe
def funcP1(pipe):
    pipe[1].send("Shchukin")
if (__name__ == '__main__'):
    pipe = Pipe()
    P1 = Process(target = funcP1, args = (pipe,))
    P1.start()
    name = pipe[0].recv()
    P1.join()
    print "name =",name
```

3) Створити додатковий процес P1 та передати з нього в головний процес список, який містить власне прізвище та номер варіанту. Приклад:

```
from multiprocessing import Process, Pipe
def funcP1(pipe):
    L = [100, "Shchukin"]
    pipe[1].send(L)
if (__name__ == '__main__'):
    pipe = Pipe()
    P1 = Process(target = funcP1, args = (pipe,))
    P1.start()
    L = pipe[0].recv()
    P1.join()
    print "variant =", L[0]
    print "name =",L[1]
```

4) Написати програму, якій головний процес відсилає додатковому процесу P1 прізвище студента, а процес P1 додає до нього ім’я студента і повертає результат, який виводиться головним процесом. Приклад:

```
from multiprocessing import Process, Pipe
def funcP1(pipe):
    name = pipe[1].recv()
    name = name + " Vladislav"
    pipe[1].send(name)
if (__name__ == '__main__'):
    pipe = Pipe()
    P1 = Process(target = funcP1, args = (pipe,))
    P1.start()
    name = "Shchukin"
    pipe[0].send(name)
    name = pipe[0].recv()
    P1.join()
    print "name =",name
```

5) Ознайомитись із прикладами лекції по моделі «клієнт-сервер». Виконати тестування наведених в лекції програм.

6) Написати розподілену програму, в якій N клієнтів C[0], …, C[N-1] взаємодіють з сервером S (рисунок 5.1). Кожен клієнт генерує випадкове ціле число q і відправляє його серверу. Сервер виконує операцію G(q) (відповідно варіанту з таблиці 5.1) і відправляє результат клієнту. Число q генерується в діапазоні [10; 20]. Прийняти N = 10.

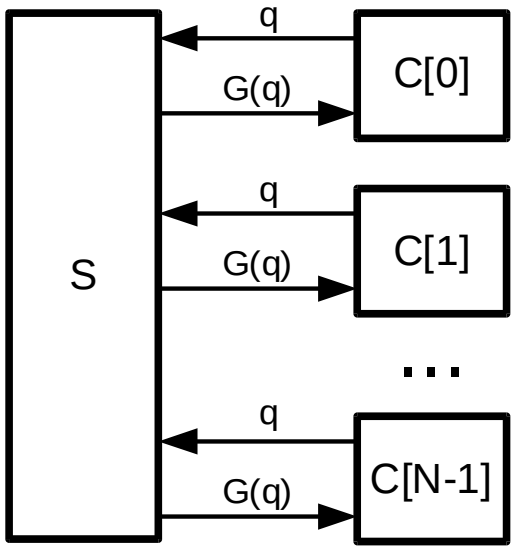



Рисунок 5.1 – Схема взаємодії клієнт-сервер

Таблиця 5.1 – Індивідуальні завдання

| **Варіант** | **Функція G(q)** | **Варіант** | **Функція G(q)** |
|---|---|---|---|
| 1 | 2*sin(q)+ln(2*q) | 11 | ctg(q)-sin(2*q) |
| 2 | q*ln(q) | 12 | 5*q*ln(q) |
| 3 | 2*cos(q) | 13 | sin(2*q)+cos(q) |
| 4 | q+tg(q) | 14 | sin(q)*(q+3) |
| 5 | ctg(q)/5 | 15 | sin(q)+ln(q) |
| 6 | 10*ln(q) | 16 | tg(q)-2*ln(q) |
| 7 | sin(q)+cos(q) | 17 | 5*cos(q)-2*q |
| 8 | tg(q)*ln(q) | 18 | 7*sin(q)+10*ln(q) |
| 9 | cos(q)-q | 19 | sin(q)+tg(3*q) |
| 10 | sin(q)/ln(q) | 20 | sin(q+5)*ln(q-2) |

7) Написати розподілену програму, в якій N серверів S[0], …, S[N-1] взаємодіють з клієнтом C (рисунок 5.2). Клієнт формує масиви X і T

кожен з яких розміром N. Кожному серверу клієнт відправляє весь масив X і один елемент T[i] (i – номер сервера). Сервер виконує операцію (відповідно варіанту з таблиці 5.2) і відправляє результат R клієнту. Елементи масивів X і T ініціалізуються випадковим цілим числом з діапазону, який вказано в таблиці 5.2. Прийняти N = 10.

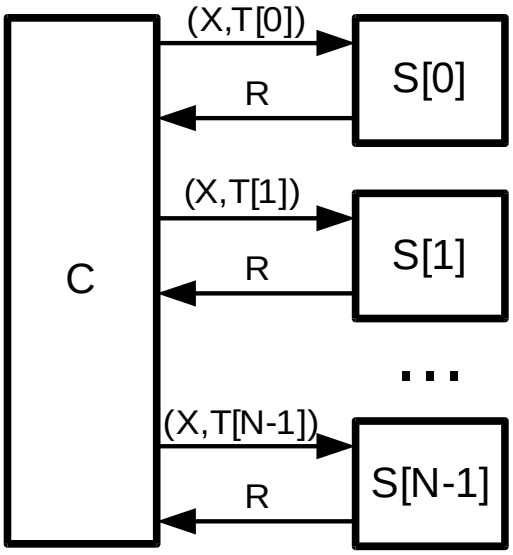



Рисунок 5.2 – Схема взаємодії клієнт-сервер

Таблиця 5.2 – Індивідуальні завдання

| **Варіант** | **Діапазон** | **Функція сервера S[i]** |
|---|---|---|
| 1 | [-10; 15] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких більше ніж T[i] |
| 2 | [5; 10] | R = F(X)+ T[i], де F – функція, яка повертає суму непарних елементів в масиві X |
| 3 | [7; 23] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких кратні значенню T[i] |
| 4 | [-17; -5] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких дорівнює T[i] |
| 5 | [-3; 20] | R = F(X) - T[i], де F – функція, яка повертає кількість непарних елементів в масиві X |

| Варіант | Діапазон | Функція сервера S[i] |
|---|---|---|
| 6 | [10; 45] | R = F(X)*T[i], де F – функція, яка повертає максимальний елемент масиву X |
| 7 | [-17; 44] | R = F(X) + T[i], де F – функція, яка повертає кількість від'ємних елементів масиву X |
| 8 | [29; 79] | R = F(X) * T[i], де F – функція, яка повертає кількість елементів масиву X, які кратні трьом |
| 9 | [-80; -20] | R = F(X) - T[i], де F – функція, яка повертає кількість елементів масиву X, які більші ніж число -35 |
| 10 | [10; 90] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких менше ніж T[i] |
| 11 | [5; 60] | R = F(X)+ T[i], де F – функція, яка повертає суму парних елементів в масиві X |
| 12 | [-150; -55] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких не кратні значенню T[i] |
| 13 | [-1; 100] | R = F(T[i], X), де F – функція, яка повертає кількість елементів в масиві X, значення яких не дорівнює T[i] |
| 14 | [22; 55] | R = F(X) - T[i], де F – функція, яка повертає кількість парних елементів в масиві X |
| 15 | [-75; 10] | R = F(X)*T[i], де F – функція, яка повертає мінімальний елемент масиву X |
| 16 | [-15; 83] | R = F(X) + T[i], де F – функція, яка повертає кількість не від'ємних елементів масиву X |
| 17 | [88; 105] | R = F(X) * T[i], де F – функція, яка повертає кількість елементів масиву X, які кратні п'яти |
| 18 | [-6; 33] | R = F(X) - T[i], де F – функція, яка повертає кількість елементів масиву X, які менші ніж число 10 |

| Варіант | Діапазон | Функція сервера S[i] |
|---|---|---|
| 19 | [28; 99] | R = F(T[i], X) + 10, де F – функція, яка повертає кількість елементів в масиві X, значення яких не дорівнює T[i] |
| 20 | [15; 29] | R = F(X) + 2*T[i], де F – функція, яка повертає кількість елементів масиву X, які більші ніж число 20 |

## Контрольні питання

1. Що таке іменований канал?
2. Як передати великий обсяг даних від одного процеса іншому?
3. Пояснити модель «клієнт-сервер».
4. Які недоліки та переваги мають багатопоточні сервери?
5. Як організувати систему типу P2P (Peer-to-peer)?

## САМОСТІЙНА РОБОТА № 6

**Тема:** реалізація конвеєрних алгоритмів.

**Мета:** ознайомитись з типами конвеєрних алгоритмів; програмно реалізувати конвеєрні алгоритми на мові Python.

### Хід роботи

1) Ознайомитись з прикладом відкритого конвеєру з лекції по конвеєрним алгоритмам. Виконати тестування програми та навести програму і результати тестування.

2) Ознайомитись з прикладом циклічного конвеєру з лекції по конвеєрним алгоритмам. Виконати тестування програми та навести програму і результати тестування.

3) Ознайомитись з прикладом закритого конвеєру з лекції по конвеєрним алгоритмам. Виконати тестування програми та навести програму і результати тестування.

4) Розглянути програму, яка представлена у прикладі нижче. Переробити її таким чином, щоб результатом виведення було власне прізвище студента. Побудувати структурну схему конвеєра. Який тип конвеєра використовується: відкритий, циклічний або закритий?

```
from multiprocessing import Process, Queue
N=8

def funcP0(q):
    t = 'S'
    q.put(t)

def funcPi(NumProcess, q_in, q_out):
    t = q_in.get()
    if (NumProcess == 1): t = t + 'h'
```

```
        elif (NumProcess == 2): t = t + 'c'
        elif (NumProcess == 3): t = t + 'h'
        elif (NumProcess == 4): t = t + 'u'
        elif (NumProcess == 5): t = t + 'k'
        elif (NumProcess == 6): t = t + 'i'
        q_out.put(t)

    if (__name__ == "__main__"):
        q = []
        for i in range(N-1):
            q.append(Queue())

        P = []
        P.append(Process(target=funcP0,     args     =
(q[0],)))
        for i in range(N-2):
            P.append(Process(target=funcPi,   args   =
(i+1,q[i],q[i+1])))

        for i in range(N-1):
            P[i].start()

        t = q[N-2].get()
        t = t + 'n'

        for i in range(N-1):
            P[i].join()

        print t
```

5) Написати розподілену програму, в якій N клієнтів P[0], …, P[N-1] об'єднані в циклічний конвеєр (рисунок 6.1). Кожен процес генерує випадкове ціле число q і відправляє його по колу наступному процесу. Після цього процеси виконують операцію G(q) (відповідно варіанту з таблиці 6.1) і записують результат у свій файл (для кожного процесу має бути створений свій власний файл). Число q генерується в діапазоні [10; 20]. Прийняти N = 8.

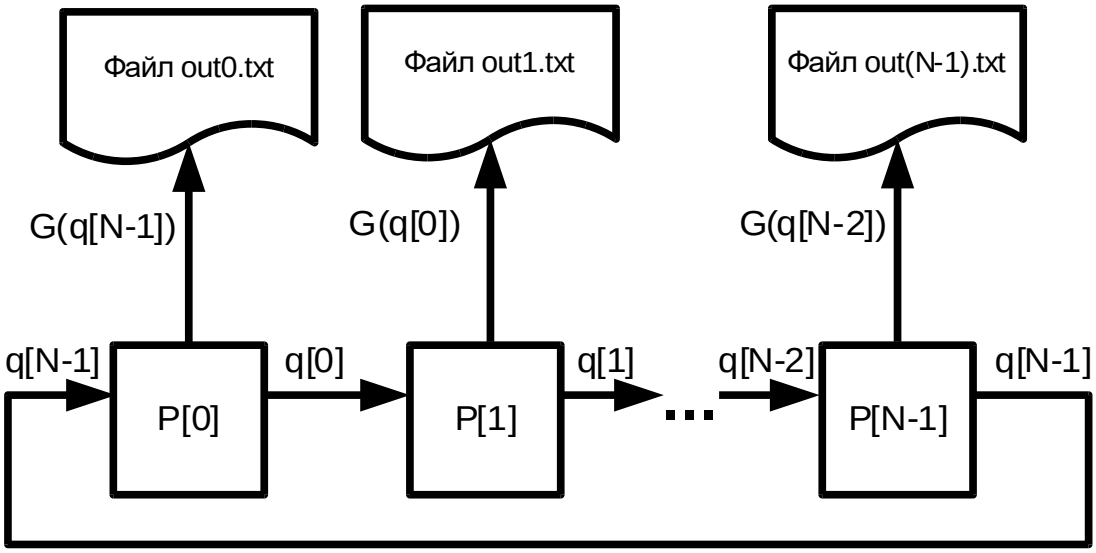



Рисунок 6.1 – Схема роботи потоків

Таблиця 6.1 – Завдання до виконання

| **Варіант** | **Функція G(q)** | **Варіант** | **Функція G(q)** |
|---|---|---|---|
| 1 | 2*sin(q)+ln(2*q) | 11 | ctg(q)-sin(2*q) |
| 2 | q*ln(q) | 12 | 5*q*ln(q) |
| 3 | 2*cos(q) | 13 | sin(2*q)+cos(q) |
| 4 | q+tg(q) | 14 | sin(q)*(q+3) |
| 5 | ctg(q)/5 | 15 | sin(q)+ln(q) |
| 6 | 10*ln(q) | 16 | tg(q)-2*ln(q) |
| 7 | sin(q)+cos(q) | 17 | 5*cos(q)-2*q |
| 8 | tg(q)*ln(q) | 18 | 7*sin(q)+10*ln(q) |
| 9 | cos(q)-q | 19 | sin(q)+tg(3*q) |
| 10 | sin(q)/ln(q) | 20 | sin(q+5)*ln(q-2) |

## Контрольні питання

1. Що таке конвеєрний алгоритм?
2. Які типи конвеєрних алгоритмів Вам відомі?
3. Перерахувати функції для роботи з файлами на мові Python.
4. Які засоби синхронізації потоків існують в Python?
5. Що таке критична секція коду?

## САМОСТІЙНА РОБОТА № 7

**Тема:** програмування сокетів на мові Python.

**Мета:** ознайомитись з типами сокетів; програмно реалізувати клієнт-серверну архітектуру з використанням сокетів на мові Python.

### Хід роботи

1) Ознайомитись з основами для роботи із сокетами (https://docs.python.org/2/howto/sockets.html).

Відповісти на питання:

Чим відрізняється TCP від UDP?

Що таке порт? Які порти є зарезервованими для системних задач?

Яка послідовність операцій для TCP-з'єднання між клієнтом та сервером (навести схему)?

Яка послідовність операцій для UDP-з'єднання між клієнтом та сервером (навести схему)?

2) Розглянути текст програми-сервера, який використовує протокол TCP. Для кожного рядка програми додати свої коментарі: коротко записати, що робить кожний рядок програми.

```
# Server (TCP)
import socket
s = socket.socket(socket.AF_INET,
  socket.SOCK_STREAM)
s.bind(("", 9999))
s.listen(1)
print "Waiting..."
client, addr = s.accept()
print str(addr)+" connect\n"
d = client.recv(1024)
print "Data from client: " + d
client.send(d + " OK")
client.close()
```

3) Розглянути текст програми-клієнта, який використовує протокол TCP, який в якості запиту передає прізвище та ініціали (необхідно змінити текст програми, щоб передавалися правильне прізвище та ініціали). Для кожного рядка програми додати свої коментарі: коротко записати, що робить кожний рядок програми.

```
# Client (TCP)
import socket
s = socket.socket(socket.AF_INET,
  socket.SOCK_STREAM)
s.connect(("127.0.0.1", 9999))
s.send("Hello World")
d = s.recv(1024)
print "Data from server: " + d
s.close()
```

3) Виконати тестування сервера та клієнта. Для цього необхідно запустити програми для клієнта і для сервера в окремому вікні командного рядка. Приклад виконання вище наведених програм:

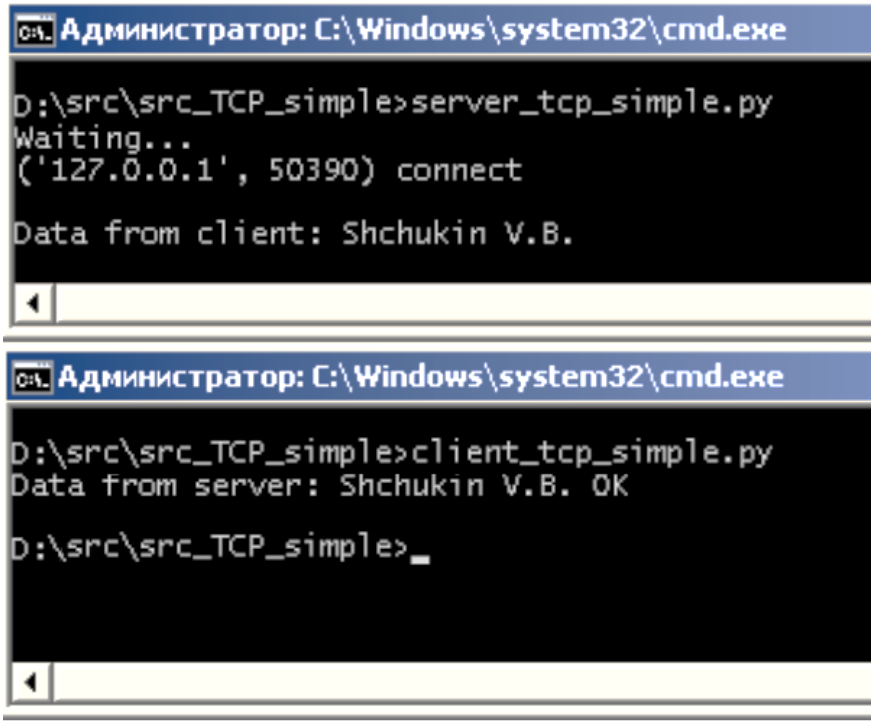



Рисунок 7.1 – Приклад роботи клієнта та сервера

4) Розглянути тексти сервера та клієнта, які використовують протокол UDP. Для кожного рядка програми додати свої коментарі: коротко записати, що робить кожний рядок програми. Виконати тестування програм, попередньо змінивши прізвище та ініціали, які передаються в програмі.

```
# Server (UDP)
import socket
s = socket.socket(socket.AF_INET,
  socket.SOCK_DGRAM)
s.bind(("", 9999))
print "Waiting..."
data, address = s.recvfrom(256)
print "Data from client: " + data
s.sendto(data + " OK", address)
s.close()
# Client (UDP)
import socket
s = socket.socket(socket.AF_INET,
  socket.SOCK_DGRAM)
s.sendto("Hello World",("127.0.0.1", 9999))
data, address = s.recvfrom(1024)
print "Data from server: " + data
s.close()
```

5) Самостійно виконати експерименти з функціями dumps та loads модуля pickle.

## Контрольні питання

1) Що таке мережний сокет?

2) Які основні принципи мережної взаємодії додатків із застосуванням портів?

3) Коротко охарактеризуйте стек протоколів TCP/IP.

4) Перерахуйте номери портів таких протоколів та служб: HTTP, DNS, FTP, Telnet, ICQ, Skype.

5) Як співвідноситься модель відкритої взаємодії OSI та стек протоколів TCP/IP?

## ТЕМИ РЕФЕРАТІВ

1. Тенденції розвитку суперкомп’ютерних обчислень. Найбільш потужні обчислювальні комплекси в світі.
2. Основи паралельних і розподілених обчислень.
3. Структури паралельних та розподілених комп’ютерних систем.
4. Топології розподілених комп’ютерних систем.
5. Поняття мультикомп’ютера, мультипроцесора, обчислювального кластера.
6. Концепція GRID-систем.
7. Класифікації Фліна та Джонсона обчислювальних комп’ютерних систем.
8. Поняття процесу та потоку. Стан процесу. Взаємодія процесів. Тупики.
9. Поняття багатозадачності та багатопоточності.
10. Концепція необмеженого паралелізму. Коефіцієнт прискорення та коефіцієнт ефективності паралельних обчислень.
11. Паралельні алгоритми. Представлення, побудова та аналіз.
12. Етапи розроблення паралельного алгоритму.
13. Паралельні алгоритми для задач лінійної алгебри (вектори, матриці).
14. Закон Амдала.
15. Рівні паралелізму в багатоядерних системах.
16. Взаємодія процесів через спільні змінні. Завдання взаємного виключення і синхронізації. Критична секція коду.
17. Засоби синхронізації: семафори, м’ютекси, монітори.
18. Взаємодія процесів через посилання повідомлень. Механізм рандеву.
19. Моделі паралельних обчислень.
20. Стратегія паралельних обчислень «Виробник-споживач».
21. Конвеєрні паралельні алгоритми.
22. Взаємоблокування потоків та методи боротьби з цим явищем.
23. Розподілені обчислення. Модель клієнт-сервер. Мережні сокети.

24. Особливості програмування для кластерних систем.
25. Бібліотеки паралельного програмування: MPI, OpenMP, Win32.
26. Програмний пакет PVM (Parallel Virtual Machine) для створення розподілених систем.
27. Життєвий цикл потоку та процесу в операційній системі.
28. Відмінності основних та фонових потоків. Іменування потоків.
29. Налаштування пріоритету потоку. Потоки реального часу. Витіснення потоку.
30. Робота с потоками мовою C# з використанням засобів .NET Framework.
31. Порівняльна характеристика потоків (thread) та процесів (process).
32. Методи блокування потоків. Використання методів Join, Sleep.
33. Конструкції блокування на мові C#: Lock, Mutex, Semaphore.
34. Сигнальні конструкції на базі класу EventWaitHandle.
35. Поняття потокової безпеки. Поняття атомарності операцій.
36. Потокобезпечні типи .NET Framework.
37. Засоби примусового розблокування потоків (Abort, Interrupt).
38. Фреймворк для розподілених обчислень Windows Communication Foundation (WCF).
39. Поняття контракту, прив’язки, адреси WCF. Хостинг служби WCF.
40. Короткий огляд технології NVIDIA CUDA для паралельних обчислень.

## СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Златопольский Д. Основы программирования на языке Python / Д. Златопольский. – М.: ДМК Пресс, 2017. – 285 с.

2. Любанович Б. Простой Python. Современный стиль программирования / Б. Любанович. – СПб.: Питер, 2017. – 480 с.

3. Лутц М. Python. Карманный справочник / М. Лутц. – М.: Вильямс, 2016. – 320 с.

4. Бизли Д. Python. Подробный справочник / Д. Бизли; пер. с англ. Киселев А. – СПб.: Символ-Плюс, 2010. – 864 с.

5. Forbes E. Learning Concurrency in Python / E. Forbes. – Birmingham: Packt Publishing, 2017. – 313 p.

6. Chou E. Mastering Python Networking / E. Chou. – Birmingham: Packt Publishing, 2017. – 446 p.

**ЗМІСТ**

М е т о д и ч н і  в к а з і в к и
до виконання самостійних робіт
з дисципліни «Паралельні та розподілені обчислення»
для студентів спеціальності
123 «Комп’ютерна інженерія»
усіх форм навчання

УКЛАДАЧІ: Музика Іван Олегович
Кузнєцов Денис Іванович

Реєстраційний № ___

Підписано до друку _____ 2021 р.
Формат _____А5_____
Обсяг _____57__ стор.
Тираж _____ прим.

Видавничий центр
Криворізького національного університету,
вул. Віталія Матусевича, 11, м. Кривий Ріг