**ЛАБОРАТОРНА РОБОТА № 1**

Процеси та потоки

**Мета заняття:** взаємодія між процесами, розподіл даних між процесами,робота з файлами які відображуються у пам’ять.

Хід роботи :

Завдання 1. Створення та заповнення випадковими числами:

Лістинг програми:

import struct  
def create\_file(filename, size):  
 with open(filename, 'wb') as f:  
 for \_ in range(size):  
 number = random.randint(10, 100) #рандом чисел від 10 до 100  
 f.write(struct.pack('i', number))#Записуємо як 4-байтове ціле число  
if \_\_name\_\_ == '\_\_main\_\_':  
 create\_file('data.dat', 30) # Створюємо файл з 30 випадковими числами  
 print("Файл 'data.dat' створено та заповнено випадковими числами.")

Результат виконання:





Рис 1 Створений файл

Генерація нових чисел та сортування:

import random  
import struct  
  
def regenerate\_file(filename, size):  
 with open(filename, 'wb') as f: # Відкриваємо файл у режимі запису в бінарному форматі  
 numbers = [] # Ініціалізуємо пустий список для збереження згенерованих чисел  
 for \_ in range(size):  
 number = random.randint(10, 100) # Генеруємо випадкові числа від 10 до 100  
 f.write(struct.pack('i', number)) # Записуємо 4-байтове ціле число у файл  
 numbers.append(number) # Додаємо згенероване число до списку  
  
 # Виводимо згенеровані числа  
 print("Нові числа, записані у файл:")  
 for i, num in enumerate(numbers, start=1):  
 print(f"{i}: {num}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 regenerate\_file('data.dat', 30) # Генеруємо та записуємо 30 нових випадкових чисел у файл

Результат виконання:

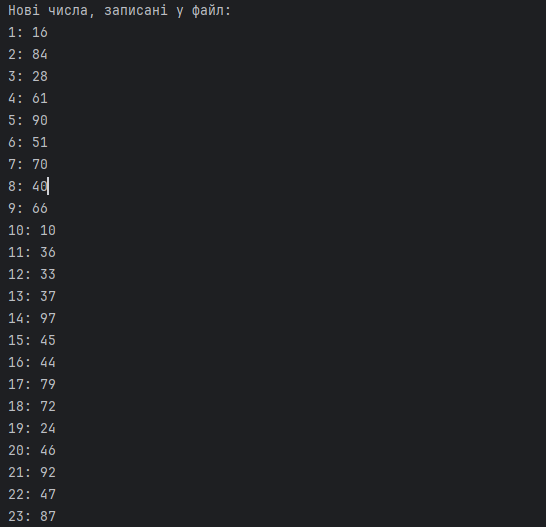


Рис 2

Сортування:

import mmap  
import struct  
  
def sort\_file(filename, size):  
 with open(filename, 'r+b') as f:  
 mm = mmap.mmap(f.fileno(), 0) # Відображаємо файл у пам'ять  
  
 # Зчитуємо дані з файлу та конвертуємо їх у масив цілих чисел  
 numbers = [struct.unpack('i', mm[i \* 4:(i + 1) \* 4])[0] for i in range(size)]  
  
 # Бульбашкове сортування  
 for i in range(size - 1):  
 for j in range(size - i - 1):  
 if numbers[j] > numbers[j + 1]:  
 numbers[j], numbers[j + 1] = numbers[j + 1], numbers[j]  
  
 # Записуємо відсортовані числа назад у файл  
 for i in range(size):  
 mm[i \* 4:(i + 1) \* 4] = struct.pack('i', numbers[i])  
  
 mm.close()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 sort\_file('data.dat', 30) # Сортуємо файл із 30 числами  
 print("Дані у файлі 'data.dat' відсортовано.")



Рис 3

Виведення:

import struct  
  
def read\_and\_display\_sorted(filename, count):  
 try:  
 with open(filename, 'rb') as f:  
 data = f.read()  
 print("Файл успішно прочитано. Відсортовані числа:")  
 # Розбираємо байти на цілі числа  
 numbers = [struct.unpack('i', data[i \* 4:(i + 1) \* 4])[0] for i in range(count)]  
 # Виводимо кожне число  
 for idx, number in enumerate(numbers, start=1):  
 print(f"{idx}: {number}")  
 except Exception as e:  
 print(f"Помилка при читанні файлу: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 read\_and\_display\_sorted('data.dat', 30) # Виводимо 30 чисел з файла

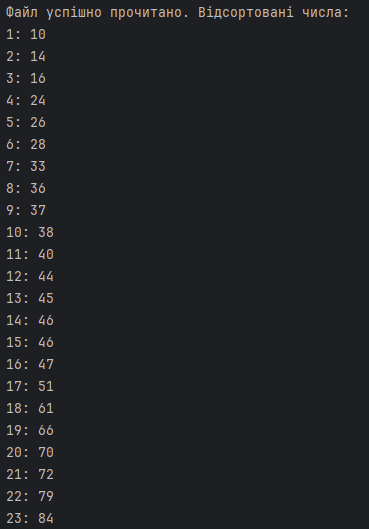


Рис 4 Відсортовані числа

Завдання 2.

import random  
import struct  
import threading  
  
lock = threading.Lock() # Об'єкт блокування для синхронізації  
  
def regenerate\_and\_display(filename, size):  
 with lock: # Критична секція, яка блокує доступ для інших потоків  
 try:  
 numbers = []  
 with open(filename, 'r+b') as f:  
 f.truncate(0) # Очищаємо вміст файлу  
 # Генеруємо нові числа  
 for \_ in range(size):  
 number = random.randint(10, 100)  
 f.write(struct.pack('i', number))  
 numbers.append(number)  
 # Виводимо числа  
 print("Нові числа:")  
 for idx, num in enumerate(numbers, start=1):  
 print(f"{idx}: {num}")  
 except Exception as e:  
 print(f"Помилка при роботі з файлом: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 regenerate\_and\_display('data.dat', 30) # Викликаємо функцію для регенерації та виведення чисел

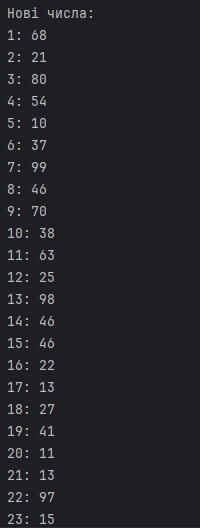


Рис 5

Сортування:

import mmap  
import struct  
import threading  
  
lock = threading.Lock() # Об'єкт блокування для синхронізації  
  
def sort\_file(filename, size):  
 with lock: # Критична секція, яка блокує доступ для інших потоків  
 try:  
 with open(filename, 'r+b') as f:  
 mm = mmap.mmap(f.fileno(), 0) # Відображаємо файл у пам'ять  
  
 # Читання даних  
 numbers = [struct.unpack('i', mm[i \* 4:(i + 1) \* 4])[0] for i in range(size)]  
  
 # Сортування бульбашкою  
 for i in range(size - 1):  
 for j in range(size - i - 1):  
 if numbers[j] > numbers[j + 1]:  
 numbers[j], numbers[j + 1] = numbers[j + 1], numbers[j]  
  
 # Запис назад  
 for i in range(size):  
 mm[i \* 4:(i + 1) \* 4] = struct.pack('i', numbers[i])  
 mm.close()  
 print("Файл 'data.dat' відсортовано.")  
 except Exception as e:  
 print(f"Помилка при сортуванні: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 sort\_file('data.dat', 30)

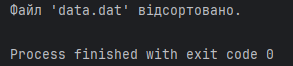


Рис 6

Вивід

import mmap  
import struct  
import threading  
  
# Створюємо об'єкт блокування для синхронізації доступу до спільного ресурсу  
lock = threading.Lock()  
  
def display\_file(filename, size):  
 with lock: # Використовуємо блокування для критичної секції  
 try:  
 with open(filename, 'rb') as f: # Відкриваємо файл для читання  
 # Відображаємо файл у пам'ять  
 mm = mmap.mmap(f.fileno(), 0, access=mmap.ACCESS\_READ)  
  
 # Зчитуємо дані  
 numbers = [struct.unpack('i', mm[i \* 4:(i + 1) \* 4])[0] for i in range(size)]  
  
 # Виводимо числа  
 print("Числа у файлі:")  
 for idx, number in enumerate(numbers, start=1):  
 print(f"{idx}: {number}")  
  
 # Закриваємо відображення у пам'ять  
 mm.close()  
 except Exception as e:  
 print(f"Помилка при читанні файлу: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 display\_file('data.dat', 30) # Виводимо 30 чисел з файлу

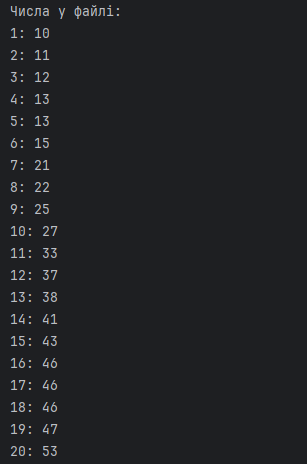


Рис 7

Сортування:

import mmap  
import struct  
import threading  
  
lock = threading.Lock() # Об'єкт блокування для синхронізації  
  
def reverse\_sort\_file(filename, size):  
 with lock: # Критична секція  
 try:  
 with open(filename, 'r+b') as f:  
 mm = mmap.mmap(f.fileno(), 0) # Відображаємо файл у пам'ять  
  
 # Читання даних  
 numbers = [struct.unpack('i', mm[i \* 4:(i + 1) \* 4])[0] for i in range(size)]  
  
 # Сортування вставками у зворотному напрямку  
 for i in range(size):  
 key = numbers[i]  
 j = i - 1  
 while j >= 0 and numbers[j] < key:  
 numbers[j + 1] = numbers[j]  
 j -= 1  
 numbers[j + 1] = key  
  
 # Запис результатів  
 for i in range(size):  
 mm[i \* 4:(i + 1) \* 4] = struct.pack('i', numbers[i])  
 mm.close() # Закриваємо відображення у пам'ять  
  
 # Вивід відсортованих чисел  
 print("Відсортовані числа у зворотному напрямку:")  
 for idx, number in enumerate(numbers, start=1):  
 print(f"{idx}: {number}")  
 except Exception as e:  
 print(f"Помилка при сортуванні: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 reverse\_sort\_file('data.dat', 30)

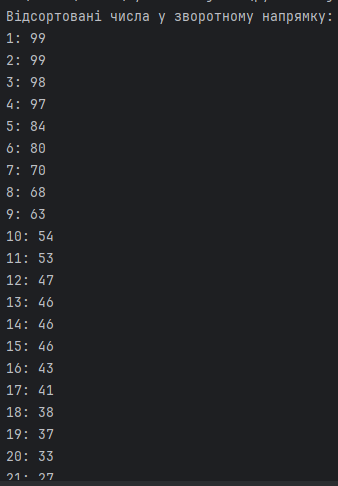


Рис 8

**Висновок:**

В цьому занятті ми досліджували взаємодію між процесами, а саме, способи розподілу даних між ними та роботу з файлами, що відображаються у пам'ять.

Один із способів взаємодії між процесами - це використання блокування для синхронізації доступу до спільного ресурсу. Ми використовували об'єкт блокування **threading.Lock()** для забезпечення того, що тільки один потік може отримувати доступ до спільного ресурсу одночасно. Це допомагає уникнути конфліктів при одночасному доступі до файлів або спільних даних.

Ми також вивчали роботу з файлами, що відображаються у пам'ять. Використовуючи модуль **mmap**, ми можемо отримати доступ до файлу як до області пам'яті. Це дає нам можливість працювати з файлами так само, як і зі звичайними байтовими рядками, але з додатковою швидкістю доступу та можливістю змінювати дані безпосередньо в пам'яті.