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:

```
Файл 'data.dat' створено та заповнено випадковими числами.
```

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:

```
Нові числа, записані у файл:
1: 16
2: 84
3: 28
4: 61
5: 90
6: 51
7: 70
8: 40
9: 66
10: 10
11: 36
12: 33
13: 37
14: 97
15: 45
16: 44
17: 79
18: 72
19: 24
20: 46
21: 92
22: 47
23: 87
```

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.")
```

```
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
```

```
Файл успішно прочитано. Відсортовані числа:
1: 10
2: 14
3: 16
4: 24
5: 26
6: 28
7: 33
8: 36
9: 37
10: 38
11: 40
12: 44
13: 45
14: 46
15: 46
16: 47
17: 51
18: 61
19: 66
20: 70
21: 72
22: 79
23: 84
```

Рис 4 Відсортовані числа

Завдання 2.

```python
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
```

```
Нові числа:
1: 68
2: 21
3: 80
4: 54
5: 10
6: 37
7: 99
8: 46
9: 70
10: 38
11: 63
12: 25
13: 98
14: 46
15: 46
16: 22
17: 13
18: 27
19: 41
20: 11
21: 13
22: 97
23: 15
```

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)
```

```
Файл 'data.dat' відсортовано.

Process finished with exit code 0
```

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
```

```
Числа у файлі:
1: 10
2: 11
3: 12
4: 13
5: 13
6: 15
7: 21
8: 22
9: 25
10: 27
11: 33
12: 37
13: 38
14: 41
15: 43
16: 46
17: 46
18: 46
19: 47
20: 53
```

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)
```

```
Відсортовані числа у зворотному напрямку:
1: 99
2: 99
3: 98
4: 97
5: 84
6: 80
7: 70
8: 68
9: 63
10: 54
11: 53
12: 47
13: 46
14: 46
15: 46
16: 43
17: 41
18: 38
19: 37
20: 33
21: 27
```

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.