Task 1,2,3,4 pdf

Name ahmed ayman elsayed.

1. ما هو الBuffer

- الـ Buffer هو مساحة مؤقتة في الذاكرة تُستخدم لتخزين البيانات أثناء نقلها بين مصدر (مثل قرص أو شبكة) ووجهة (مثل معالج أو ملف).

- الهدف هو تقليل التفاعل المباشر مع العمليات البطيئة (مثل القراءة/الكتابة على القرص).

2. كيف يعمل في الكود؟

بدون Buffer: يتم جلب كل عنصر ومعالجته فورًا. هذا قد يكون بطيئًا إذا كان جلب البيانات أو معالجتها يتطلب وقتًا (مثل I/O).

- مع Buffer: يتم جمع البيانات في حاوية (vector) حتى تمتلئ (حجم الـ Buffer)، ثم تُعالج دفعة واحدة. هذا يقلل من عدد العمليات المباشرة.

3. تأثير الأداء:

- المزايا:

  - تقليل الوصول المتكرر للمصادر البطيئة (مثل القرص أو الشبكة).

  - تحسين الكفاءة عند معالجة البيانات دفعة واحدة بدلاً من عنصر ب عنصر.

- العيوب:

  - زيادة استهلاك الذاكرة بسبب تخزين البيانات مؤقتًا.

  - تأخير طفيف في المعالجة حتى يمتلئ الـ Buffer.

النتيجة المتوقعة:

عند تشغيل الكود، ستلاحظ أن الوقت المستغرق مع الـ Buffer قد يكون أقل في حال كانت عملية `fetchData` أو `processData` بطيئة (في الواقع، مثل قراءة ملف). في هذا المثال البسيط، الفرق قد لا يكون واضحًا لأن العمليات خفيفة، لكن الفكرة تنطبق على سيناريوهات أكثر تعقيدًا.

```
#include <iostream>
#include <vector>
#include <chrono> // For time measurement
using namespace std;

// Simulate a data source (e.g., reading from a file or network)
int fetchData(int index) {
    return index; // Simple simulation: return the index as data
}

// Process the data (e.g., writing to a file or displaying)
void processData(int data) {
    cout << "Processed: " << data << endl;
}

// Implementation without Buffer
void withoutBuffer(int size) {
    cout << "Without Buffer:" << endl;
    auto start = chrono::high_resolution_clock::now();

    for (int i = 0; i < size; i++) {
        int data = fetchData(i); // Fetch data
        processData(data);       // Process immediately
    }

    auto end = chrono::high_resolution_clock::now();
```

```
    auto duration = chrono::duration_cast<chrono::milliseconds>(end - start);
    cout << "Time without Buffer: " << duration.count() << " milliseconds" << endl;
}

// Implementation with Buffer
void withBuffer(int size, int bufferSize) {
    cout << "\nWith Buffer:" << endl;
    vector<int> buffer; // The buffer
    buffer.reserve(bufferSize); // Pre-allocate space

    auto start = chrono::high_resolution_clock::now();

    for (int i = 0; i < size; i++) {
        buffer.push_back(fetchData(i)); // Store data in the buffer

        // When the buffer is full, process the data
        if (buffer.size() == bufferSize) {
            for (int data : buffer) {
                processData(data);
            }
            buffer.clear(); // Empty the buffer
        }
    }

    // Process any remaining data in the buffer
    for (int data : buffer) {
        processData(data);
```

```
    }

    auto end = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(end - start);
    cout << "Time with Buffer: " << duration.count() << " milliseconds" << endl;
}

int main() {
    int dataSize = 20;      // Total data size
    int bufferSize = 5;     // Buffer size

    withoutBuffer(dataSize); // Run without Buffer
    withBuffer(dataSize, bufferSize); // Run with Buffer

    return 0;
}
```