24.Apr 2023 Misato Seki

USING SEMAPHORES

1. INTRODUCTION

In this report, we will write a program that requires 'synchronization' using semaphores and observe how it is processed.

2. WHAT IS THIS CODE DOING?

This program shows how two threads (t1 and t2) can be used to prepare an array called myVec and process it synchronously.

The first thread (t1) executes the prepareWork function. This function inserts four elements into the myVec array and releases the prepareSignal semaphore, indicating that the preparation is complete. prepareSignal semaphore has only one slot, which is initialized to 0, so when the release() function is called the value of the slot is set to 1 and the waiting thread is released.

The second thread (t2) executes the completeWork function. This function modifies the second element of the array. It waits for the prepareSignal semaphore, and when the semaphore is released, it modifies the myVec element to indicate that the operation is complete.

When both threads exit using the join() function, all elements of myVec are displayed after they have been modified.

Thus, this program demonstrates how the two threads are synchronized and work cooperatively using the same vectors. (PICTURE. 1)

3. WHAT WE LEARNED

When using a Critical Section such as the myVec array in this example, the concept of "synchronization" is important. In other words, to maintain the integrity of the Critical Section, it is necessary to control that only one thread accesses the Critical Section and waits for the next thread that wants to access it.

.

Appendix

```cpp
//g++ -std=c++20
#include <iostream>
#include <semaphore>
#include <thread>
#include <vector>

std::vector<int> myVec{};

// the value of prepareSignal is set '0'.
std::counting_semaphore<1> prepareSignal(0);

void prepareWork() {

    //insert 4 items into myVec.
    myVec.insert(myVec.end(), {0, 1, 0, 3});
    std::cout << "Sender: Data prepared."  << '\n';
    // rekease prepareSignal semapho which means the preparation has done.
    prepareSignal.release();
    // release() function sets the value to 1.
}

void completeWork() {

    std::cout << "Waiter: Waiting for data." << '\n';
    // waiting the value becoming 0 -> 1.
    prepareSignal.acquire();
    myVec[2] = 2;
    std::cout << "Waiter: Complete the work." << '\n';
    for (auto i: myVec) std::cout << i << " ";
    std::cout << '\n';

}

int main() {

    std::cout << '\n';

    std::thread t1(prepareWork);
    std::thread t2(completeWork);

    t1.join();
    t2.join();

    std::cout << '\n';
misatoseki@MisatonoMacBook-Air Session16 % ./threadSynchronizationSemaphore.out

Waiter: Waiting for data.
Sender: Data prepared.
Waiter: Complete the work.
0 1 2 3
```

Picture1. code and executed result.