Lec-31

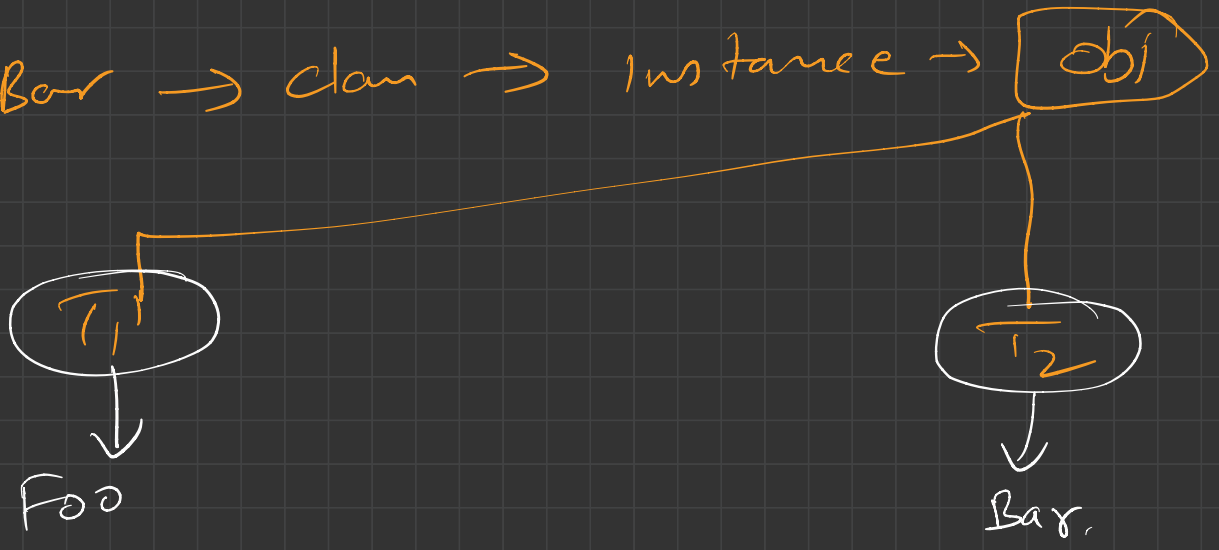
Leetcode concurrency

3

1. Print FooBar
2. Print zero Even odd
3. Building H2O ☆

① Print → FooBar

FooBar → class → instance → obj



Single Process / thread → Foo Bar Foo Bar.

Two threads → $T_1 \rightarrow$ $T_2 \rightarrow$ $T_1 \rightarrow$ $T_2 \rightarrow$

Code =

Watch Lec-23

```cpp
class FooBar {
private:
    int n;
    std:: mutex m;
    std::condition_variable cv;
    bool turn;
public:
    FooBar(int n) {
        this->n = n;
        turn = 0;
    }

    void foo(function<void()> printFoo) {

        for (int i = 0; i < n; i++) {
            std::unique_lock<std::mutex> lock(m);
            while(turn == 1){
                cv.wait(lock);
            }
            printFoo();
            turn = 1;
            cv.notify_all();
        }
    }

    void bar(function<void()> printBar) {

        for (int i = 0; i < n; i++) {
            std::unique_lock<std::mutex> lock(m);
            while(turn == 0){
                cv.wait(lock);
            }
            printBar();
            turn = 0;
            cv.notify_all();
        }
    }
};
```

Initially turn = 0

T_1

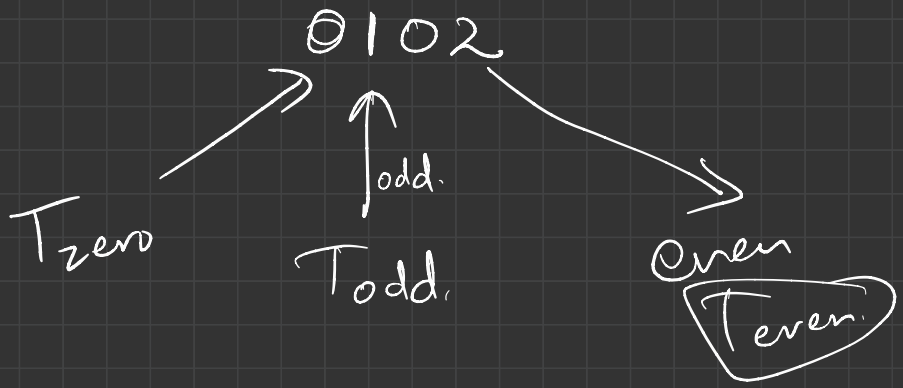
turn = 1

T_2

turn = 0

Problem 2 Print zero Even odd

=> n = 2



Solution 1

n = 3 → 0 1 0 2 0 3

T_zero T_odd T_even

n = 2

```
class ZeroEvenOdd {
private:
    int n;
    std:: mutex m;
    std::condition_variable cv;
    int turn;
    int i;
public:
    ZeroEvenOdd(int n) {
        this->n = n;
        turn = 0;
        i = 1;
    }

    // printNumber(x) outputs "x", where x is an inte
    void zero(function<void(int)> printNumber) {
        while(i<=n){
            std::unique_lock<std::mutex> lock(m);
            while(turn != 0 && i <= n){
                cv.wait(lock);
            }
            if(i>n){
                break;
            }
            printNumber(0);
            turn = (i % 2) == 0 ? 2 : 1;
            cv.notify_all();
        }
    }
```

Turn = 0

```
    void even(function<void(int)> printNumber) {
        while(i<=n){
            std::unique_lock<std::mutex> lock(m);
            while(turn != 2 && i <= n){
                cv.wait(lock);
            }
            if(i>n){
                break;
            }
            printNumber(i++);
            turn = 0;
            cv.notify_all();
        }
    }

    void odd(function<void(int)> printNumber) {
        while(i<=n){
            std::unique_lock<std::mutex> lock(m);
            while(turn != 1 && i <= n){
                cv.wait(lock);
            }
            if(i>n){
                break;
            }
            printNumber(i++);
            turn = 0;
            cv.notify_all();
        }
    }
};
```

i = 3 turn = 2

turn = 1

Case 1: n = 2

① i = 1 → turn = 1

② i = 2 → turn = 2

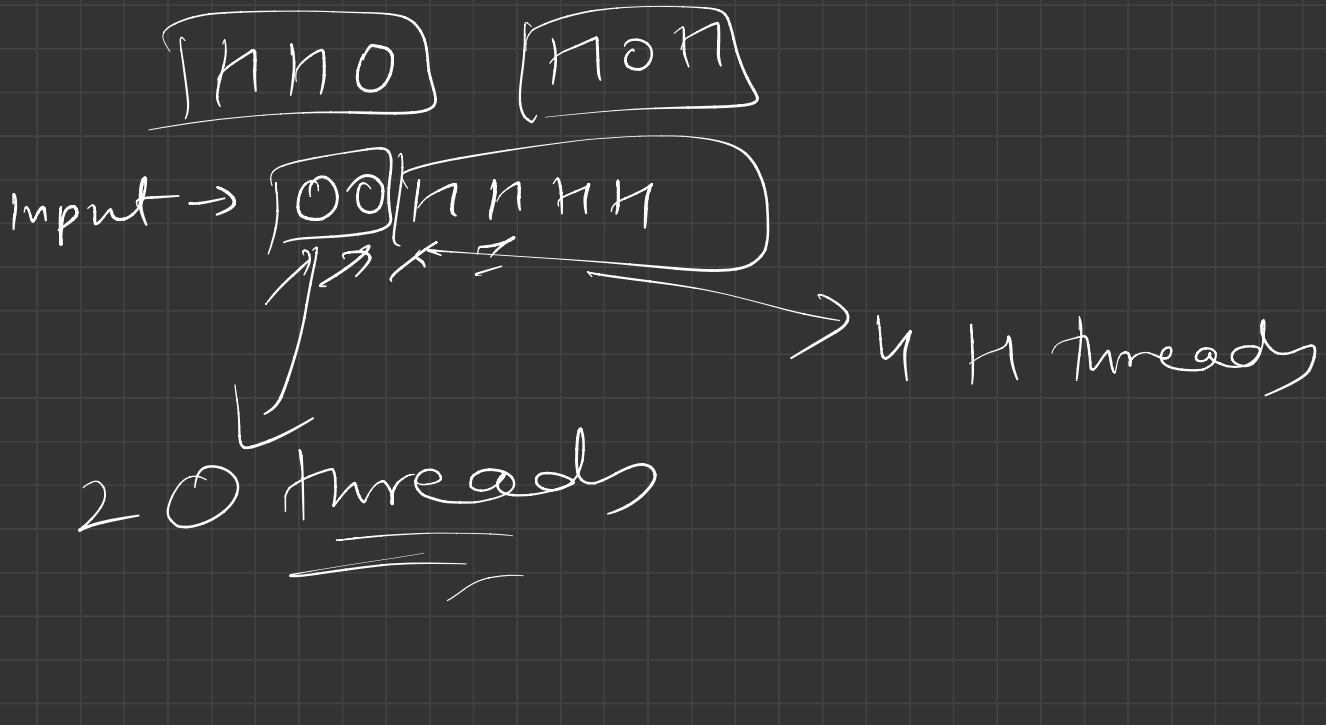
③ i = 3

output: 0102

(3) H2O

=> H → Threads multiple

O → Threads → multiple

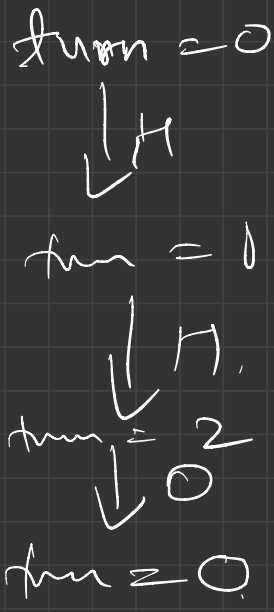


solution:-

```
class H2O {
    std:: mutex m;
    std::condition_variable cv;
    int turn;
public:
    H2O() {
       turn = 0;
    }

    void hydrogen(function<void()> releaseHydrogen) {
        std::unique_lock<std::mutex> lock(m);
        while(turn == 2){
            cv.wait(lock);
        }
        releaseHydrogen();
        ++turn;
        cv.notify_all();
    }

    void oxygen(function<void()> releaseOxygen) {
        std::unique_lock<std::mutex> lock(m);
        while(turn < 2){
            cv.wait(lock);
        }
        releaseOxygen();
        turn = 0;
        cv.notify_all();
    }
};
```



Condvars!

turn →

mutex

Conditons