* 3 Problems

Print in Order:

Suppose we have a class:

public class Foo {

public void first() { print("first"); }

public void second() { print("second"); }

public void third() { print("third"); }

}

The same instance of Foo will be passed to three different threads. Thread A will call first(), thread B will call second(), and thread C will call third(). Design a mechanism and modify the program to ensure that second() is executed after first(), and third() is executed after second().

Soln :

class Foo {

std::mutex m;

std::condition\_variable cv;

int turn;

public:

Foo() {

turn = 0;

}

void first(function<void()> printFirst) {

// printFirst() outputs "first". Do not change or remove this line.

printFirst();

turn = 1;

cv.notify\_all();

}

void second(function<void()> printSecond) {

// printSecond() outputs "second". Do not change or remove this line.

std::unique\_lock<std::mutex> lock(m);

while(turn!=1){

cv.wait(lock);

}

printSecond();

turn =2;

cv.notify\_all();

}

void third(function<void()> printThird) {

// printThird() outputs "third". Do not change or remove this line.

std::unique\_lock<std::mutex> lock(m);

while(turn!=2){

cv.wait(lock);

}

printThird();

}

};

2. Fizz Buzz problem

You have the four functions:

* printFizz that prints the word "fizz" to the console,
* printBuzz that prints the word "buzz" to the console,
* printFizzBuzz that prints the word "fizzbuzz" to the console, and
* printNumber that prints a given integer to the console.

You are given an instance of the class FizzBuzz that has four functions: fizz, buzz, fizzbuzz and number. The same instance of FizzBuzz will be passed to four different threads:

* **Thread A:** calls fizz() that should output the word "fizz".
* **Thread B:** calls buzz() that should output the word "buzz".
* **Thread C:** calls fizzbuzz() that should output the word "fizzbuzz".
* **Thread D:** calls number() that should only output the integers.

Modify the given class to output the series [1, 2, "fizz", 4, "buzz", ...] where the ith token (**1-indexed**) of the series is:

* "fizzbuzz" if i is divisible by 3 and 5,
* "fizz" if i is divisible by 3 and not 5,
* "buzz" if i is divisible by 5 and not 3, or
* i if i is not divisible by 3 or 5.

Soln:

class FizzBuzz {

private:

int n;

mutex m;

condition\_variable c;

int i;

public:

FizzBuzz(int n) {

this->n = n;

this->i = 1;

}

// printFizz() outputs "fizz".

void fizz(function<void()> printFizz) {

while(i<=n){

unique\_lock<mutex> lock(m);

while(i<=n && ((i%3==0) && (i%5!=0))==0){

c.wait(lock);

}

if(i<=n){

printFizz();

i++;

}

c.notify\_all();

}

}

// printBuzz() outputs "buzz".

void buzz(function<void()> printBuzz) {

while(i<=n){

unique\_lock<mutex> lock(m);

while(i<=n && ((i%3!=0) && (i%5==0))==0){

c.wait(lock);

}

if(i<=n){

printBuzz();

i++;

}

c.notify\_all();

}

}

// printFizzBuzz() outputs "fizzbuzz".

void fizzbuzz(function<void()> printFizzBuzz) {

while(i<=n){

unique\_lock<mutex> lock(m);

while(i<=n && ((i%3==0) && (i%5==0))==0){

c.wait(lock);

}

if(i<=n){

printFizzBuzz();

i++;

}

c.notify\_all();

}

}

// printNumber(x) outputs "x", where x is an integer.

void number(function<void(int)> printNumber) {

while(i<=n){

unique\_lock<mutex> lock(m);

while(i<=n && ((i%3!=0) && (i%5!=0))==0){

c.wait(lock);

}

if(i<=n){

printNumber(i);

i++;

}

c.notify\_all();

}

}

};

Dining Philosopher leetcode problem:

https://leetcode.com/problems/the-dining-philosophers/