Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Операційні системи

Лабораторна робота 1
“Взаємодія між задачами”

Виконав студент 3-го курсу

групи ІПС-32

Євчик Олексій Юрійович

2023

Варіант

Мова реалізації - C++,

Засіб комунікації - перенаправлений ввід/вивід,

Спосіб синхронізації - неблокуючий ввід-вивід,

"носій" - обчислення - процес,

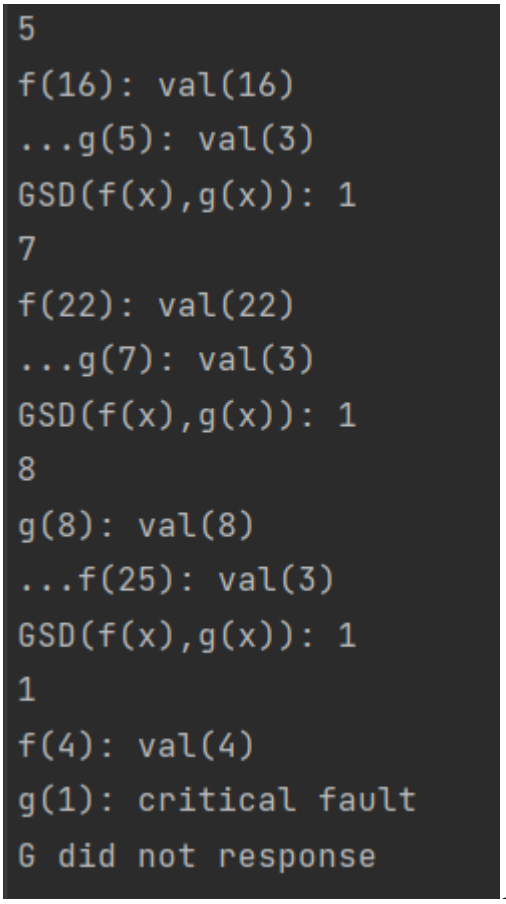
бінарна операція - gcd

Реалізація

https://github.com/MoloZzz/OS_lab1v2/tree/master

Робота

Приклад способу синхронізації(неблокуючий ввід вивід, або по іншому синхронне виконання функцій):



Як ми бачимо, виконуються функції синхронно, якщо G повертає значення раніше то його і виводить раніше, якщо ж F то вона виводиться першою.

```
0
f(0): val(24)
g(0): val(24)
GSD(f(x),g(x)): 24
6
g(6): val(6)
...f(19): val(3)
GSD(f(x),g(x)): 3
7
f(22): val(22)
...g(7): val(3)
GSD(f(x),g(x)): 1
```

Якщо ж одночасно, то виводить то одну то іншу

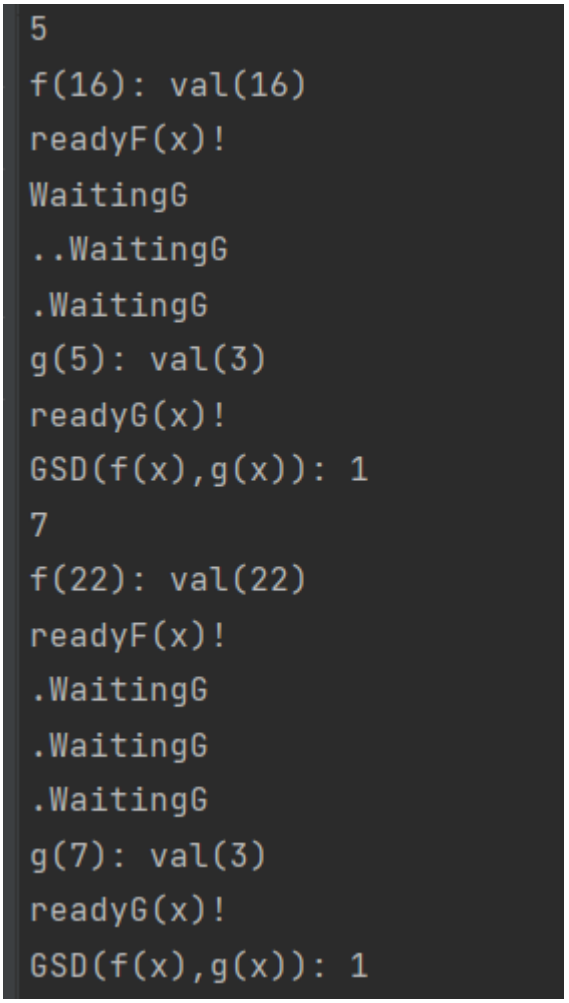
```
0
g(0): val(24)
f(0): val(24)
GSD(f(x),g(x)): 24
```

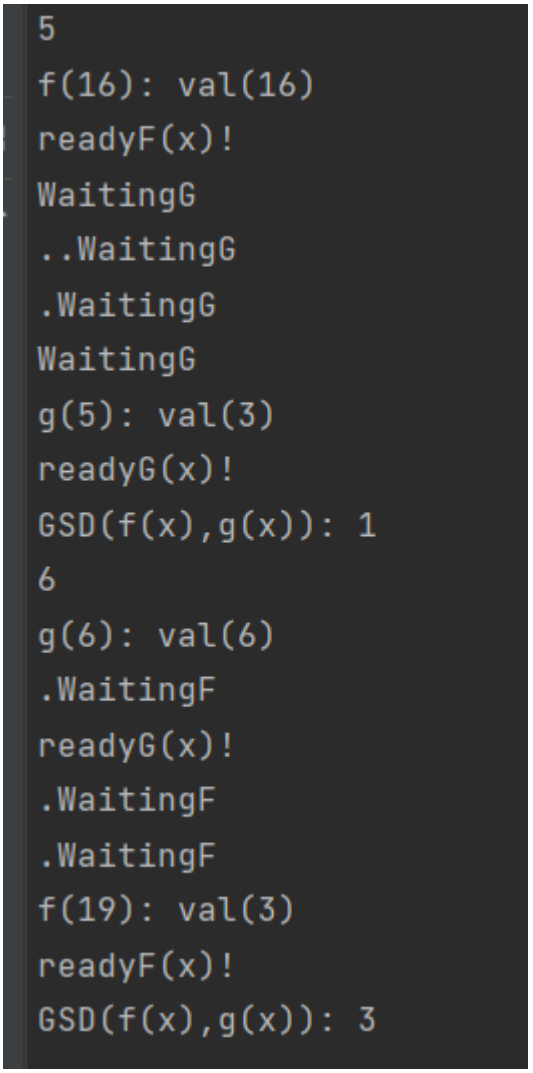
"носій" - обчислення - процес. Виконання функцій відбувається на одному потоці, використовуючи механізм асинхронного виконання функцій

```
auto fResult : future<…> = std::async( policy: std::launch::async, &: f, x);

auto gResult : future<…> = std::async( policy: std::launch::async, &: g, x);

shortDT resF = fResult.get();
shortDT resG = gResult.get();
```

Використовуючи стандартну бібліотеку “future” `#include <future>`

Я додав окремий вивід менеджера, аби бачити, що функції f(), g(), manager() працюють синхронно. Я додав окремий вивід менеджера, аби переконатись, в їх синхронній роботі:





Функції f та g можуть повернути **значення** або критичну **помилку**. Якщо ж функція(compfunc(x) трохи перероблений шаблон) повертає soft_fault то обчислення продовжується, поки не отримаємо результат або поки не вийде час виділений функції. (по закінченню роботи функції - сама функція виводить значення в консоль, окремо менеджер повідомляє, що функція повернула результат). Якщо ж функція **не закінчить свою роботу за певний час**(я установив цей час, як 4 секунди), то менеджер повинен зупинити очікування та **kill process**. Сам алгоритм дій прописаний, проте виникли складності в зупинці процеса.

```
auto result : comp_result<unsigned int> = os::lab1::compfunc::compfunc(x);

while (std::holds_alternative<os::lab1::compfunc::soft_fault>( v: result)) {
    std::cout << "." << std::flush;
    result = os::lab1::compfunc::compfunc(x);
}

std::cout <<"f(" << x << "): " << result << std::endl;

return result;
```

Тайм Аут:

```
    if(std::chrono::steady_clock::now() - start_time >= timeout){
        std::cout << "time out" << std::endl;
        timeoutFlag = true;
        break;
    }
} while (
        (statusF != std::future_status::ready || statusG != std::future_status::ready));


if(timeoutFlag){
    std::cout<< "Timeout. Calculation Failed" << std::endl;
    //kill f g
    break;
}else{
    resF = fResult.get();
    resG = gResult.get();
}
```

Якщо ж час перевищив ліміт, то ми не продовжуємо надалі перевіряти статус виконання функцій, а просто зупиняємо процеси та повертаємо помилку обчислень(в коді процес не зупиняється, тому що я не можу це реалізувати).

Мемоізація

або ж збереження результату, для подальшого використання в разі потреби, реалізовано за допомогою кешу.

В моїй реалізації, результати вже виконаних обчислень зберігаються та повторно використовуються при наступних викликах з тими ж самими параметрами.

Реалізовано:

Визначається кеш для зберігання результатів функцій. Змінено функції f та g на перевірку кешу перед виконанням обчислень. Якщо результат вже збережений у кеші, повертати його, інакше викликати функцію та зберігати результат у кеші.

Я задав кеш глобально

```
std::unordered_map<int, shortDT> memo_cache;
```

Додав його обробку в основну фунцію

```
shortDT memoized_compfunc(int x) {
    auto it : iterator<...> = memo_cache.find(x);
    if (it != memo_cache.end()) {
        return it->second;
    }

    auto result : comp_result<unsigned int> = os::lab1::compfunc::compfunc(x);

    while (std::holds_alternative<os::lab1::compfunc::soft_fault>( v: result)) {
        std::cout << "." << std::flush;
        result = os::lab1::compfunc::compfunc(x);
    }

    memo_cache[x] = result;
    return result;
}
```

Шукаємо в кеші результат, якщо немає то викликаємо функцію. Окремо виніс варіант з soft_fault, як окремий випадок і зберігаю результат після останнього обрахунку(отримання результату). Тому що, якщо функція поверне кілька разів некритичну помилку, то вийде, що ми отримуємо цикл, а не реальне(потрібне) значення обчислень.

Отже, **Загальний алгоритм**:

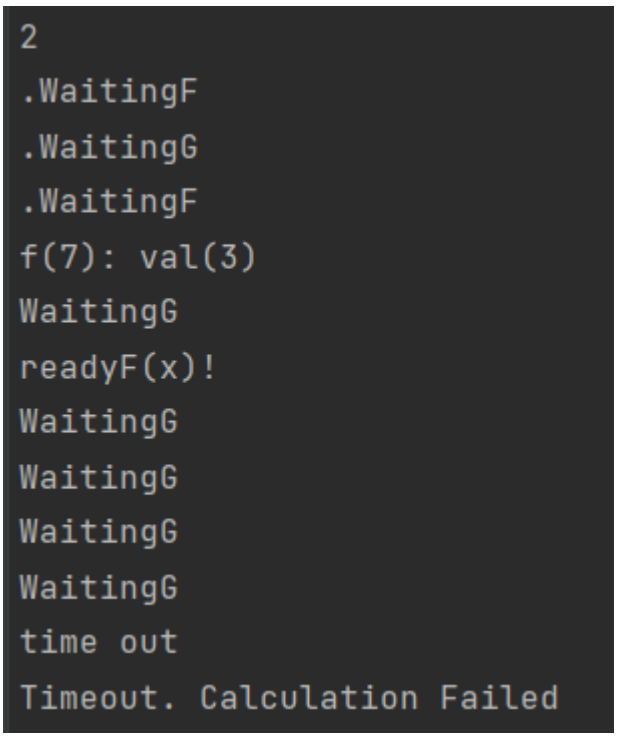
1) Запускається Менеджер
2) Користувач вводить якесь значення x
3) Менеджер запускає 2 асинхронні функції f(x) та g(x)
4) Починається виконання функцій, якщо ж compfunc повертає soft_fault то продовжуємо обчислення
   Можливі варіанти відповіді функцій f, g -
   Критичний збій, результат(int).
5) Менеджер кожні 0.5 секунди перевіряє статуси роботи функцій, якщо виконана то виводить відповідне повідомлення та змінює флаг function_done на true, та очікує на завершення другої функції, паралельно виводить повідомлення “Waiting”
6) Після отримання результату функцій, якщо обидві функції повернули значення то обраховуємо найбільший спільний дільник(gcd), Якщо ж, хоча б,одна функція повернула помилку(або вийшов час) то виводимо повідомлення про причину неможливості обрахунку бінарної функції(F/G did not response або Time out)
7) Робота менеджера закінчується, коли користувач вводить -1.

Кілька скріншотів:

```
1
f(4): val(4)
readyF(x)!
WaitingG
g(1): critical fault
readyG(x)!
Calculation failed. Function G(x) did not response
```

```
6
g(6): val(6)
.WaitingF
readyG(x)!
.WaitingF
.WaitingF
f(19): val(3)
readyF(x)!
GSD(f(x),g(x)): 3
```

```
10
.WaitingF
g(10): critical fault
readyG(x)!
.WaitingF
.f(31): val(3)
WaitingF
readyF(x)!
Calculation failed. Function G(x) did not response
```



Висновок:

Під час виконання цієї лабораторної роботи, я дізнався про нові способи міжпроцесової комунікації, такі як іменовані канали, windows-повідомлення, спільна пам’ять. Один із них, я спробував реалізувати. А також навчився синхронному виконанню функцій в C++, дізнався про примітиви взаємного виключення, та реалізував асинхронне виконання менеджера та 2 функцій за допомогою бібліотеки future та медотів std::async. Навчився використовувати кілька методів бібліотеки chrono, для контролю часу. Також дізнався про такі поняття як дедлок, mutex, та про взаємодію процесів та потоків. Можна зробити висновок, що ця сфера є надзвичайно обширною та складною в розумінні, та потребує багато часу на освоєння.