НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии
Дисциплина: «Архитектура вычислительных систем»

Многопоточное консольное приложение, моделирующее работу программистов в соответствии с заданным условием (см п.1).

Пояснительная записка

Выполнил:
Темирханов Михаил,
студент гр. БПИ199.

13.12.2020г.

Москва
2020

Содержание

1. Текст задания

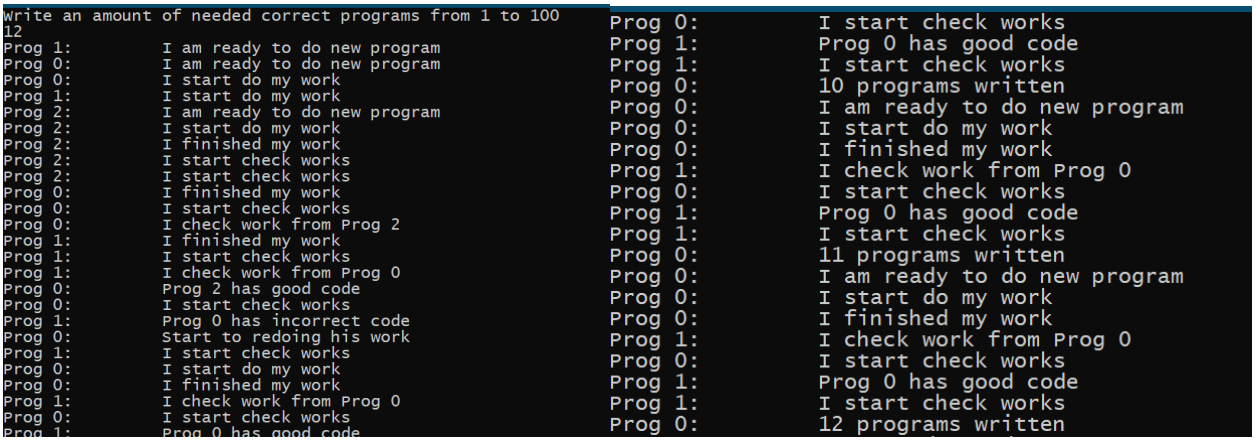
В отделе работают три программиста. Каждый программист пишет свою программу и отдает ее на проверку другому программисту. Программист проверяет чужую программу, когда его собственная уже написана. По завершении проверки, программист дает ответ: программа написана правильно или написана неправильно. Программист спит, если не пишет свою программу и не проверяет чужую программу. Программист просыпается, когда получает заключение от другого программиста. Если программа признана правильной, программист пишет другую программу, если программа признана неправильной, программист исправляет ее и отправляет на проверку тому же программисту, который ее проверял. Создать многопоточное приложение, моделирующее работу программистов

2. Описание решения

Для решения поставленной задачи было решено использовать следующий алгоритм:
Пользователем вводится число, которое характеризует число верных программ, которые 3 программиста должны написать. Программисты начинают работу и сообщают о том, что они начали писать код. После написания кода, программисты сдают его на проверку свободным программистам (тем, кто не пишет код и не проверяет чей-либо код), и заявляют о том, что готовы проверять чужой код. После проверки чужого кода, программисты проверяют, проверили ли их код, если нет – они садятся за проверку чужого кода, иначе они принимают результаты проверки и проверяют, корректно ли был написан их код. Если код был некорректным, программист переписывает программу, после чего повторяется вышеописанный цикл. Если код был корректным, в переменную, отвечающую за общее количество проверенных работ, добавляется единица, после чего программист садится за написание нового кода. Когда общее количество работ становится равным тому значению, которое задал пользователь с клавиатуры, программа завершает свою работу. На протяжении всех вышеописанных процессов в консоль выводится основная информация о статусе программистов и о итогах проверок

3. Тестирование программы

Тест 1:



Рисунки 1, 2

Тест 2:

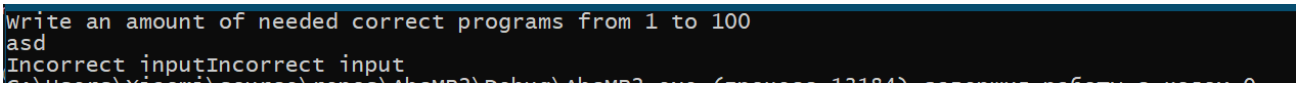


Рисунок 3

Тест 3:

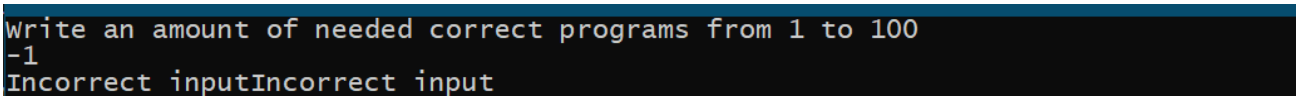


Рисунок 4

ПРИЛОЖЕНИЕ 1

Список литературы

1. [http://softcraft.ru/] (12.12.2020)
2. [https://metanit.com/cpp/tutorial/] (13.12.2020)
3. [https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160&viewFallbackFrom=vs-2019] (13.12.2020)
4. [https://proginfo.ru/processes-and-threads/] (13.12.2020)
5. [https://habr.com/ru/post/182610/] (13.12.2020)

ПРИЛОЖЕНИЕ 2

Код программы

```
#include <iostream>
#include <mutex>
#include <condition_variable>
#include <semaphore.h>
#include <vector>
#include <thread>
#include <random>

#pragma ide diagnostic ignored "EndlessLoop"

bool letsCheck = false; //Флаг проверки
std::vector<bool> isReady;
std::vector<bool> isCorrect;
std::condition_variable checkWork; //Условная переменная регулировки проверки работ
std::mutex checkMtx; //Мьютекс для проверки работ

int countOfProgramms = 0; //Количество созданных программ

const int minCodeTime = 100; //Минимальное время написания программы
const int maxCodeTime = 500; //Максимальное время написания программы
const int minCheckTime = 100; //Минимальное время проверки программы
const int maxCheckTime = 300; //Максимальное время проверки программы

/**
 * Находит индекс непроверенной работы и в случае
 * если его нет выводит -1
 * @param myIndex
 * @return
 */
int findNotCheckWork(int myIndex) {
    for (int i = 0; i < 3; ++i) {
        if (isReady[i] && !isCorrect[i] && i != myIndex)
            return i;
    }
    return -1;
}

/**
 * Реализует поток программиста
 * @param index индекс программиста
 */
void programmer(int index) {
    srand(time(0) * index); //Устанавливаем рандом
    std::unique_lock<std::mutex> checkLock(checkMtx); //Локер для условной переменной
    checkLock.unlock(); //Разлочиваем созданный локер
    while (true) {
        std::printf("Prog %d: \tI am ready to do new program\n", index);
        //Устанавливаем флаги готовности и корректности написанной программы
        isReady[index] = false;
        isCorrect[index] = false;

        while (!isCorrect[index]) { //Пока программа не корректна пытаемся ее сделать
            std::printf("Prog %d: \tI start do my work\n", index);
            //Имитируем работу программиста
            std::this_thread::sleep_for(std::chrono::milliseconds(std::rand() %
                                (maxCodeTime - minCodeTime) + minCodeTime));
```

```
            std::printf("Prog %d: \tI finished my work\n", index);

            isReady[index] = true; //Устанавливаем флаг готовности работы
            checkWork.notify_one(); //Говорим проверяющему, что появилась работа на проверку
            letsCheck = true; //Устанавливаем флаг проверки в true, чтобы проверящий вышел
из цикла
            std::this_thread::sleep_for(std::chrono::milliseconds(10)); //Даем проверяющему
время для выхода

            while (isReady[index] && !isCorrect[index]) { //Цикл проверки (Проверяем пока не
пришли резы проги)
                std::printf("Prog %d: \tI start check works\n", index);
                //Цикл устраняющий случайное пробуждение потока
                while (!letsCheck && !isCorrect[index] && isReady[index])
                    checkWork.wait(checkLock);
                letsCheck = false; //Устанавливаем флаг проверки в false

                int checkedIndex = findNotCheckWork(index); //Находим индекс непроверенной
работы
                //Начниаем проверку если наша работа готова, но не проверена
                if (checkedIndex != -1 && !isCorrect[index] && isReady[index]) {
                    std::printf("Prog %d: \tI check work from Prog %d\n", index,
checkedIndex);
                    std::this_thread::sleep_for(std::chrono::milliseconds(std::rand() %
                                                                          (maxCheckTime -
minCheckTime) + minCodeTime));
                    bool correct = std::rand() % 2 == 0; //Выносим вердикт
                    if (!correct) {
                        isReady[checkedIndex] = false;
                        std::printf("Prog %d: \tProg %d has incorrect code\n", index,
checkedIndex);
                        std::printf("Prog %d: \tStart to redoing his work\n", checkedIndex);
                    } else {
                        std::printf("Prog %d: \tProg %d has good code\n", index,
checkedIndex);
                    }
                    isCorrect[checkedIndex] = correct; //Устанавливаем флаг корректности в
полученное выше значение
                    checkWork.notify_all();
                }
            }
        }
        countOfProgramms++; //Увеличиваем количество написанных программ на 1
        std::printf("Prog %d: \t%d programs written\n", index, countOfProgramms);
    }
}

/**
 * Считывает число
 * @param minValue минимальное значение
 * @param maxValue максимальное значение
 * @return считанное число
 */
int readNumber(int minValue, int maxValue) {
    std::printf("Write an amount of needed correct programs from 1 to 100\n");
    int number;
    std::cin >> number;
    if (number < minValue || number > maxValue) {
        std::cout << "Incorrect input";
        return -1;
    }
```

```
    return number;
}

int main() {
    int temp = readNumber(1, 100); //Считываем количество программ, которое нужно написать
    if (temp == -1) {
        std::printf("Incorrect input");
        return 0;
    }
    int countProgs = temp;
    std::thread* threads = new std::thread[3]; //Создаем потоки программистов
    for (int i = 0; i < 3; ++i) {
        isReady.push_back(false);
        isCorrect.push_back(false);
        threads[i] = std::thread(programmer, i);
    }

    while (countOfProgramms < countProgs) //Ждем пока программисты напишут нужное количество
программ
        std::this_thread::sleep_for(std::chrono::milliseconds(100));

    for (int i = 0; i < 3; ++i) { // Выводим потоки из бесконечного и удаляем их
        threads[i].detach();
    }

    delete[] threads;
    return 0;
}
```