ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

Домашняя работа №4 по учебному курсу "Архитектура вычислительных систем"

Титульный лист

Вариант 19

Исполнитель:
студент группы БПИ191 А.А.Новоселов

Москва 2020

1 Текст задания вариант 19

У одной очень привлекательной студентки есть N поклонников.Традиционно в день св. Валентина очень привлекательная студенткапроводит романтический вечер с одним из поклонников. Счастливыйизбранник заранее не известен. С утра очень привлекательная студенткаполучает N «валентинок» с различными вариантами романтического вечера.Выбрав наиболее заманчивое предложение, студентка извещает счастливчикао своем согласии, а остальных – об отказе. Требуется создать многопоточноеприложение, моделирующее поведение студентки. При решенииииспользовать парадигму «клиент-сервер» с активным ожиданием. Использовать библиотеку OpenMP

2 Решение задачи

Я додумал некоторые моменты условия задачи, т.к. они не описаны явно. Например, студентка выбирает поклонника по кол-ву латинских букв(не важно заглавных или строчных) в его записке, а сама записка является строкой. Так же я не понял как извещать потоки(т.е. передавать какие то данные не через буффер) о чем-то, например об отказе студентки, поэтому я запоминал в глобальной переменной уникальный номер студент с лучшей запиской. После завершения потока сервера(т.е. когда он обработает все валентинки) каждый клиент(поклонник) сверяет свой номер с номером лучшей валентинки и выводит сообщение в консоль, если его номер совпал. Предложенный вами вариант для прошлого дз с массивом значений true/false кажется мне не очень правильным, т.к. при вычислении наиболее лучшей валентинки нужно все равно запоминать наиболее длинную и или проходить по этому массиву, или в конце в него записывать один true. В обоих случаях это лишние действия.

Программа построена на основе клиент-серверной модели поведения, где несколько потоков выдают запросы серверу, а сервер ожидает их и отвечает. Важным при решении было реализация активного ожидания для сервера. Большая часть программы разделена на 2 функции(которые выполняют создаваемые потоки): клиент и сервер. Так же есть 2 вспомогательные функции. В основной программе(функции main) считываются входные данные, создаются клиентские и серверный поток.

Для создания активного ожидания сервер проходит по вектору строк и ждет пока кто-нибудь запишет туда новую валентинку. После ее обработки она заменяется в массиве на пустую.

Для обмена данными между потоками использовались 2 переменные(буфферы) – номер студента и вектор, куда каждый добавляет свое послание. Клиент записывает – сервер только читает.

Так же для удобства тестирования в программе есть функция генерации случайной строки(из символов с ascii кодом от 32 до 126) случайной длины из диапазона [5;15].

3 Текст программы

```
#define _rand(min, max) ( rand() % ((max) - (min) + 1) + (min) ) //возвращает случайное число из диапазона

std::vector<string> valentines;
std::string buffer = ""; //текущая валентинка
int best_valentives_note = -1; // номер лучшей валентинки
int best_valentines_note_length = -1; //лучшая длина валентинки
omp_lock_t lck_buffer;
bool flag = false;

std::string generateString()
{
    //генерирует рандомную строку рандомной длины. Начальное значение задается перед вызовом(в клиенте)
    int length = _rand(5, 15);
    std::string message = "";
    for (int i = 0; i < length; ++i)
    {
        message += (char)_rand(32, 126);
    }
    return message;
}
```

```
bool check_valentines_note(string valentine)
{
    int count = 0;
    for (int i = 0; i < buffer.length(); ++i)
    {

        //выбираем только символы
        if ((valentine[i] > 64 && valentine[i] < 91) || (valentine[i] > 96 && valentine[i] < 123)) //определим входит ли буква в латинский алфавит
        {
            ++count;
        }
    }

    //узнаем подходит ли больше нам эта строка
    if (count > best_valentines_note_length)
    {
        best_valentines_note_length = count;
        return true;
    }
    return false;
}
```

```cpp
void client(int num) {
    {
        srand(static_cast<unsigned int>(int(time(NULL)) ^ num));

        //сигналом что студентка прочитала валентинку, является свободный лок
        while (!omp_test_lock(&lck_buffer))
        {
            //skip
        }

        valentines[num] = generateString(); //сгенерируем строку
        std::cout << "I m student number " << num + 1 << " and my message is " << valentines[num] << std::endl; //выведем сгенерированное сообщение

        omp_unset_lock(&lck_buffer);

        //Ждем пока студентка прочитает все валентинки
        while (!flag) {
        }

        if (best_valentives_note == num)
        {
            std::cout << "I am student " << num + 1 << " and i won the competition!" << std::endl;
        }
        //Если нет, то грустно плачем в одиночестве(
    }
}
```

```cpp
void server(int param)
{
    int i = 0;
    while (i != param)
    {
        int position = 0;

        //студенты одноврменно обавляют в конец вектора свои валентинки, после прочтения они стираются,
        //поэтому проверяем что в массиве есть хотя бы 1 непрочитанная валентинка
        for (; position < param; ++position)
        {
            if (valentines[position] != "")
            {
                break;
            }
        }

        //если в векторе еще нет валентинок(дошли до конца) то подождем еще
        if (position == param)
        {
            continue;
        }

        //пытаемся заблокировать лок
        while (!omp_test_lock(&lck_buffer)) {}

        buffer = valentines[position]; //считаем валентинку
        cout << "Server thread read message " << buffer << endl;

        //обработаем полученную валентинку
        if (check_valentines_note(buffer))
        {
            best_valentives_note = position;
        }
        valentines[position] = ""; //уберем ее из списка ожидания

        omp_unset_lock(&lck_buffer); //снимем лок, т.е. обозначим что закончили работу с общими данными

        i++;
    }
    flag = true; //сообщим другим потокам что все прочитано
}
```

```cpp
int main(int argc, char* argv[])
{
    //считаем кол-во студентов
    int number_of_students;
    std::cout << "Enter the number of admirers:" << std::endl;
    std::cin >> number_of_students;
    if (number_of_students <= 0)
    {
        std::cout << "Incorrect data! Try again" << std::endl;
        return 0;
    }

    srand(static_cast<unsigned int>(time(NULL))); //зададим случайное начальное значение
    std::cout << "A student has " << number_of_students << " admires" << std::endl;
    valentines = vector<string>(number_of_students);

    omp_init_lock(&lck_buffer); //создадим лок, чтобы 2 потока одноврменно не читали из буффера

#pragma omp parallel shared(lck_buffer, number_of_students) num_threads (number_of_students+1)
    {
        //запусстим n потоков для клиента
#pragma omp for nowait
        for (int i = 0; i < number_of_students; ++i)
        {
            client(i);
        }

        //запустим 1 поток для сервера
#pragma omp single nowait
        {
            server(number_of_students);
        }
    }

    omp_destroy_lock(&lck_buffer);
}
```