Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Департамент

Программная инженерия

Самостоятельная работа по дисциплине

«Архитектура вычислительных систем»

Тема работы: Задача о каннибалах. Племя из n дикарей ест вместе из большого горшка, который вмещает m кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей.

Выполнил: студент группы БПИ191(1) Бен Мустафа Анас Риадович.

Преподаватель: Легалов Александр Иванович

Москва 2020

Вариант 5

Задача о каннибалах. Племя из *n* дикарей ест вместе из большого горшка, который вмещает *m* кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей.

1. **Описание принципа построения работы программы.**

В данной задаче, использующей многопоточность, достаточно большое количество данных (переменных) попадало под категорию тех, доступ к которым должен осуществляться только одним потоком для избежания ошибок при выполнении программы. Поэтому было решено создать отдельную функцию *EatProcess*, содержащую в себе критическую секцию, в которой данные обрабатывал бы только один поток. Для ограничений критической секции были использованы mutex-ы.

1.1. Описание метода *EatProcess*.

Метод *EatProcess* используется как метод, в котором осуществляется вся основная логика работы программы. Метод работает следующим образом: при входе в тело метода объявлена блокировка, что гарантирует вхождение только одного потока для работы с данными в критической секции.
Далее поток вычисляет порядковый номер текущего дикаря, декрементирует переменную горшка, хранящую в себе текущее количество кусков в горшке, выводит необходимую информацию на экран пользователя.
Затем, идёт проверка на опустошенность горшка. В случае, если горшок ещё не пуст, данный поток завершает свою работу в функции и передаёт управление следующему потоку, ожидающему разблокировки входа в критическую секцию.
В случае, если на момент работы текущего потока горшок оказывается пустым, текущий поток вызывает другой поток — поток повара, ожидая завершения его работы. Поток повара, в свою очередь, наполняет горшок необходимым количеством кусков миссионера (задаётся пользователем при старте работы программы) и завершает свою работу с соответствующим сообщением, передав управление обратно потоку-вызывателю.

```
void EatProcess() {
    mtx.lock(); // Начало критической секции
        currentIndexOfDikar %= quantityOfDikari; // Порядковый номер дикаря
        currentquantityOfKuski -= 1; // Дикарь ест и из горшка вычитается один кусок

        cout << "Дикарь номер " << currentIndexOfDikar + 1 << " съел один кусок. Осталось кусков в горшке : " << currentquantityOfKuski << "
        currentIndexOfDikar += 1;

        if (currentquantityOfKuski == 0) { // В случае, если горшок опустел, вызывает поток для повара
            auto *th = new thread([&]() {
                currentquantityOfKuski = quantityOfKuski; // Повар заполняет горшок обратно
                cout << "Повар только что проснулся и заполнил горшок " << currentquantityOfKuski << " кусками миссионера. ID потока : " <<
            });
            th->join();
            delete th; // Очищаем память от использованного потока повара
        }
    mtx.unlock(); // Конец критической секции
}
```

1.2. Описание метода *repeatEatProcess.*

Метод *repeatEatProcess* используется для удобства ограничения количества итераций заполнения/опустошения горшка (пользователь задаёт количество итераций при старте работы с программой). Для того, чтобы программа не выполнялась бесконечное количество раз, на входе пользователь задаёт количество итераций (кругов) для дикарей.

```
// Функция реализована для удобства ограничения количества итераций заполнения/опустошения горшка (пользователь задаёт кол-во итераций в программе)
void repeatEatProcess() {
    for (int loopIterator = 0; loopIterator < quantityOfRepeats; ++loopIterator) {
        for (int i = 0; i < quantityOfDikari; ++i) {
            threads[i] = thread(EatProcess);
        }
        for (int i = 0; i < quantityOfDikari; ++i) {
            threads[i].join();
        }
    }
}
```

2. **Описание входных данных.**

Для удобства использования программы пользователю предлагается выбор для входных данных при запуске программы, о чём пользователь получает соответствующую информацию:

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант :
```

Важно отметить, что любые некорректные данные обрабатываются программой:

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант : 3
Неправильный ввод! Перезапустите программу и попробуйте снова!
Process finished with exit code 0
```

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант : дцфвдц
Неправильный ввод! Перезапустите программу и попробуйте снова!
Process finished with exit code 0
```

2.1. Данные могут быть переданы в программу вручную пользователем. При этом ему необходимо ввести количество повторений алгоритма, количество дикарей и количество кусков в горшке.

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант : 1

Введите количество повторений алгоритма : 2
Введите количество дикарей : 2
Введите количество кусков в горшке : 4
```

2.2. Также пользователь может воспользоваться встроенным генератором данных. При выборе этого варианта работы программы, количество повторений алгоритма, количество дикарей и количество кусков в горшке заполняются случайно сгенерированными числами из функции *getRandomNumber*.

```
// Генерируем рандомное число между значениями min и max.
int getRandomNumber(int min, int max)
{
    static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
    // Равномерно распределяем рандомное число в нашем диапазоне
    return static_cast<int>(rand() * fraction * (max - min + 1) + min);
}
```

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант : 2

Сгенерированные данные следующие :
Количество повторений алгоритма : 1
Количество дикарей : 7
Количество кусков в горшке : 38
```

3. **Описание выходных данных.**

Вне зависимости от типа входных данных (вручную или текстовым документом), в качестве выходных данных программы пользователь получает полный процесс работы алгоритма с выводами информации по работе каждого потока и ID потока, в котором данное действие произошло. Также в конце программы пользователь получает информацию о конечном количестве времени, которое было затрачено на работу программы.

```
/Users/anasbenmustafa/CLionProjects/Microproject2_threads/cmake-build-debug/testproj2
1. Ввести параметры вручную.
2. Сгенерировать параметры.
Выберете нужный вариант : 1

Введите количество повторений алгоритма : 4
Введите количество дикарей : 4
Введите количество кусков в горшке : 5

Дикарь номер 1 съел один кусок. Осталось кусков в горшке : 4. ID потока : 0x70000e5b5000
Дикарь номер 2 съел один кусок. Осталось кусков в горшке : 3. ID потока : 0x70000e532000
Дикарь номер 3 съел один кусок. Осталось кусков в горшке : 2. ID потока : 0x70000e638000
Дикарь номер 4 съел один кусок. Осталось кусков в горшке : 1. ID потока : 0x70000e6bb000
Дикарь номер 1 съел один кусок. Осталось кусков в горшке : 0. ID потока : 0x70000e532000
Повар только что проснулся и заполнил горшок 5 кусками миссионера. ID потока : 0x70000e73e000

Дикарь номер 2 съел один кусок. Осталось кусков в горшке : 4. ID потока : 0x70000e5b5000
Дикарь номер 3 съел один кусок. Осталось кусков в горшке : 3. ID потока : 0x70000e638000
Дикарь номер 4 съел один кусок. Осталось кусков в горшке : 2. ID потока : 0x70000e6bb000
Дикарь номер 1 съел один кусок. Осталось кусков в горшке : 1. ID потока : 0x70000e532000
Дикарь номер 2 съел один кусок. Осталось кусков в горшке : 0. ID потока : 0x70000e638000
Повар только что проснулся и заполнил горшок 5 кусками миссионера. ID потока : 0x70000e73e000

Дикарь номер 3 съел один кусок. Осталось кусков в горшке : 4. ID потока : 0x70000e6bb000
Дикарь номер 4 съел один кусок. Осталось кусков в горшке : 3. ID потока : 0x70000e5b5000
Дикарь номер 1 съел один кусок. Осталось кусков в горшке : 2. ID потока : 0x70000e532000
Дикарь номер 2 съел один кусок. Осталось кусков в горшке : 1. ID потока : 0x70000e5b5000
Дикарь номер 3 съел один кусок. Осталось кусков в горшке : 0. ID потока : 0x70000e638000
Повар только что проснулся и заполнил горшок 5 кусками миссионера. ID потока : 0x70000e532000

Дикарь номер 4 съел один кусок. Осталось кусков в горшке : 4. ID потока : 0x70000e6bb000

Конечное время работы программы : 1 ms
```

Список используемых источников

1. https://habr.com/ru/post/72929/

2. https://www.rsdn.org/article/baseserv/critsec.xml

3. https://ru.bmstu.wiki/Критическая_секция

4. http://cppstudio.com/post/339/