*Московский Государственный Технический Университет имени Н. Э. Баумана.*

*ФАКУЛЬТЕТ “Информатики и систему управления”*

*КАФЕДРА “Программное обеспечение ЭВМ и информационные технологии”*

Отчет

По лабораторной работе №8

По курсу “Анализ Алгоритмов”

Тема “Поточный алгоритм”

Студент: Бадалян Д.А.

Группа: ИУ7-51

Преподаватель: Волкова Л.Л.

Москва, 2017

**Постановка задачи:**

1. Реализовать любой поточный алгоритм
2. На основе этого, а также предыдущих лабораторных работ по этой теме - сделать выводы о параллельном программировании в целом, как о способе решения задач

**Описание выбранного алгоритма и его реализации**

Выбор пал на реализацию обычного веб-сервера, точнее на имитацию работы веб-сервера. Язык программирования C#.

Моделируется стандартная модель “клиент-сервер”, где необходима реализация следующего функционала:

1. осуществление ввода-вывода пользователя(ей), в качестве вводимой информации – номер запрашиваемой веб-страницы, в случае нашей имитации – номер структуры из массива структур
2. выдача информации о странице, которая была запрошена. В случае реальных веб-серверов – если страница не находится в кэше, чтение с диска может занять приличное время. В нашей имитации кроме выдачи страницы в поток вставлен цикл, имитирующий работу
3. фоновые вычисления. В нашей имитации фоновой задачей будет подсчет суммы полей структуры. Аналогично с 2) будет вставлен некий цикл, имитирующий работу

Однопоточная реализация:

При такой реализации как минимум может возникать ситуация, когда CPU простаивает в ожидании операции ввода-вывода пользователя. Так же могут возникать ситуации, в которых фоновые вычисления, которые по сути уже не фоновые, т.к. речь идет о однопоточной реализации, выполняются слишком долго, а к серверу подключается другой пользователь и хочет ввести свой запрос. Последняя из перечисленных проблем решается грамотным планированием, но число выдаваемых страниц в минуту уменьшается существенно. О простаивании процессора во время ввода-вывода пользователя говорить не приходится.

Многопоточная реализация:

Поток №1 – диспетчер. Получает номер запрашиваемой страницы и передает его потоку №2, который осуществляет “считывание с диска”, и потоку №3, осуществляющему некие расчеты. В итоге поток №1 производит минимальные по времени выполнения операции и опять возвращается к пользователю. При условии большого времени ожидания ввода-вывода так же все будет в порядке, т.к. другие процессы параллельно будут производить необходимые вычисления/операции считывания. Можно было реализовать более гибкую схему, при которой потоки №2 и №3 могли выполнять как считывание, так и вычисления. Т.е. диспетчер отдает необходимую задачу первому свободному потоку, а не тому, который за это отвечает. Но для целей данной лабораторной работы алгоритма описанного выше алгоритма достаточно.

**Листинг**

class data

{

public int page\_ID;

public int all\_views;

public int AVG\_views;

public int last\_veiw;

public int users;

public data(int ID, int v1, int v2, int v3, int count)

{

page\_ID = ID;

all\_views = v1;

AVG\_views = v2;

last\_veiw = v3;

users = count;

}

public void print()

{

Console.WriteLine("\t\tПоток №2 - Работа закончена: web site info:");

Console.WriteLine("\t\tПоток №2 - Работа закончена: ID = {0}, views by all time = {1}, AVG day views = {2}",

page\_ID, all\_views, AVG\_views);

Console.WriteLine("\t\tПоток №2 - Работа закончена: last view time = {0}, user count = {1}", last\_veiw, users);

}

public void some\_calculations()

{

Console.WriteLine("\t\t\t Поток №3 - Работа закончена: Сумма просмотров = " + (all\_views + AVG\_views + last\_veiw));

}

}

class Program

{

static Queue<int> queue1 = new Queue<int>();

static Queue<int> queue2 = new Queue<int>();

static Thread t1 = new Thread(f1), t2 = new Thread(f2), t3 = new Thread(f3);

static List<data> db = new List<data>();

static int input\_command, work\_emulation = 1000000;

static bool work = true;

static void f1()

{

while (work)

{

string input;

Console.WriteLine("Введите номер веб-страницы этого веб-сервера, они от 1 до 7, или 0 для завершения");

input = Console.ReadLine();

input\_command = Convert.ToInt32(input);

if (input\_command == 0)

{

Console.WriteLine("Завершение работы...");

work = false;

}

else

{

if (input\_command > 7 || input\_command < 0)

Console.WriteLine("Ошибка ввода, попробуйте еще раз");

else

{

queue1.Enqueue(input\_command);

queue2.Enqueue(input\_command);

}

}

}

}

static void f2()

{

int index,temp;

while (work)

{

if (queue1.Count != 0)

{

index = queue1.Dequeue();

Console.WriteLine("\t\tПоток №2 Работаю...");

for (int i = 0; i < work\_emulation; i++)

{

//Some work

}

db[index - 1].print();

}

}

}

static void f3()

{

int index, temp;

while (work)

{

if (queue2.Count != 0)

{

Console.WriteLine("\t\t\tПоток №3 Работаю...");

for (int i = 0; i < work\_emulation; i++)

{

//Some work

}

index = queue2.Dequeue();

db[index - 1].some\_calculations();

}

}

}

Пояснения к листингу:

data - информация запрашиваемая у веб-сервера

queue1 – очередь заданий потока, отвечающего за считывание информации о веб-странице из диска

queue2 – очередь заданий потока, отвечающего за некие вычисления

В потоках 2 и 3 имитируется некая работа, в результате которой поток не выводит результат своей работы мгновенно.

**Вывод**

1. Высоконагруженный веб-сервер, реализованный на одном потоке невозможен
2. Параллельное программирование – способ решения задач, который активно применяется в своей области и в некоторых задачах является необходимым, что было выявлено в результате данной лабораторной работы. Однако существуют задачи, распараллеливание которых невозможно. Так же стоит заметить, что в любой задаче, распараллеливание которой возможно, любое программирование, не являющееся параллельным – априори неэффективно и может применяться только в случаях, когда производительность не играет большой роли. Но, не лишним будет сказать, что параллельное программирование требует другой квалификации от программиста, целью получения которой и являлись данные лабораторные работы.