Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Уфимский государственный авиационный технический университет

Кафедра вычислительной математики и кибернетики

РАБОЧАЯ ТЕТРАДЬ

для выполнения лабораторных работ

по дисциплине «Системы реального времени»

по направлениям подготовки

09.03.04 «Программная инженерия»

Букатина Виктория Викторовна ПРО-411

Кабирова Лилия Рустамовна ПРО-411

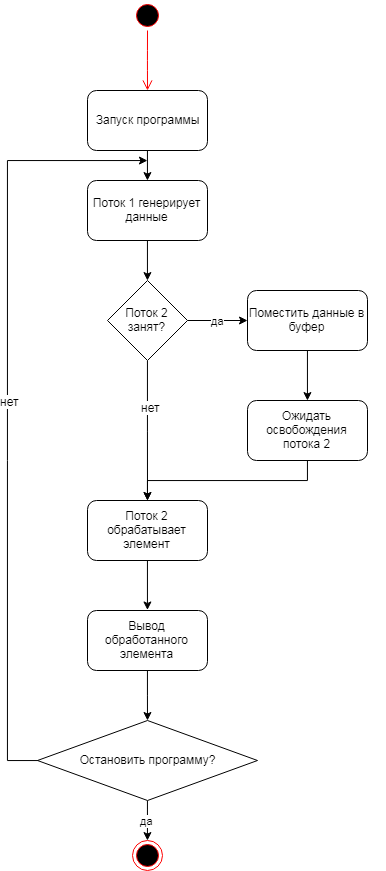
Уфа – 2018

**Лабораторная работа № 1**

Задача «Производитель-Потребитель» (Producer-Consumer)

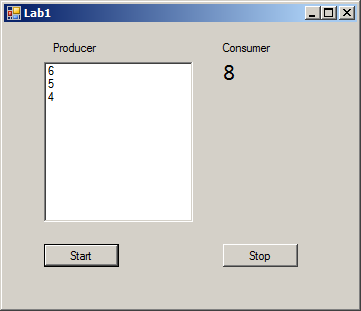
**Задание:** разработать программу одна часть которой генерирует некоторые сообщения, другая определённым образом на них реагирует

**Описание выполнения работы:**



Для реализации использовался язык программирования C#, так как он поддерживает параллельное выполнение кода через многопоточность. В C# довольно просто описываются потоки, а также запуск созданных потоков не вызывает трудностей.

Скриншот работы программы:



**Листинг программы:**

namespace SRV1

{

public partial class Form1 : Form

{

Mutex mutex;

Queue<int> queue;

bool work = true;

public Form1()

{

InitializeComponent();

r = new Random();

t1 = new Thread(new ThreadStart(tr1));

t2 = new Thread(new ThreadStart(tr2));

mutex = new Mutex();

queue = new Queue<int>();

}

Random r;

int GetNum()

{

return r.Next(1, 9);

}

void tr1()

{

while (work)

{

int num = GetNum();

mutex.WaitOne();

queue.Enqueue(num);

int[] arr= new int[queue.Count];

queue.CopyTo(arr, 0);

lbBuf.Invoke(new Action(() => lbBuf.Items.Add(num.ToString())));

mutex.ReleaseMutex();

Thread.Sleep(r.Next(1000, 2000));

}

}

void tr2()

{

while (work)

{

mutex.WaitOne();

bool has\_elem = queue.Count>0;

int num=0;

if (has\_elem)

{

num = queue.Dequeue();

lbBuf.Invoke(new Action(() => lbBuf.Items.RemoveAt(0)));

}

mutex.ReleaseMutex();

lblNow.BeginInvoke(new Action(() => lblNow.Text = (has\_elem ? num.ToString() : "")));

Thread.Sleep(r.Next(1500, 2500));

}

}

Thread t1;

Thread t2;

private void btnStart\_Click(object sender, EventArgs e)

{

t1.Start();

t2.Start();

}

private void btnStop\_Click(object sender, EventArgs e)

{

work = false;

Thread.Sleep(300);

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

work = false;

Thread.Sleep(2500);

}

}

}

**Список используемых источников:**

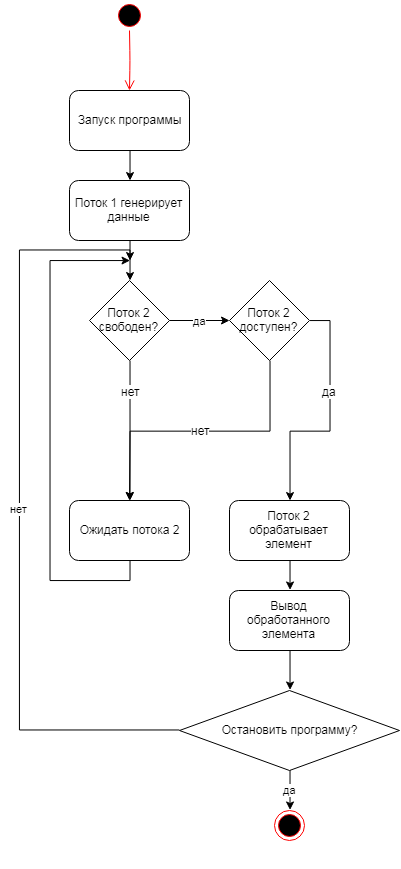
1. <https://metanit.com/sharp/tutorial/11.2.php>
2. <https://habr.com/post/126495/>
3. <https://vscode.ru/prog-lessons/potoki-v-si-sharp.htm>

**Лабораторная работа № 2**

«Пул потоков (Thread Pool)»

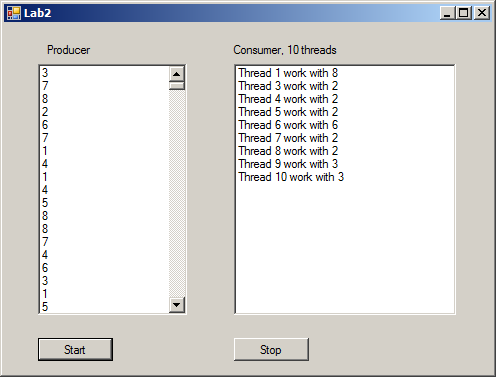
**Задание:** Реализовать пул потоков для усовершенствования решения лабораторной работы № 1.

**Описание выполнения работы:**



Для реализации использовался язык программирования C#, так как он поддерживает параллельное выполнение кода через многопоточность. В C# довольно просто описываются потоки, а также запуск созданных потоков не вызывает трудностей.

Скриншот работы программы:



**Листинг программы:**

namespace SRV2

{

public partial class Form1 : Form

{

Mutex mutex;

Queue<int> queue;

bool work = true;

Semaphore semaphore;

public Form1()

{

InitializeComponent();

r = new Random();

t1 = new Thread(new ThreadStart(tr1));

mutex = new Mutex();

queue = new Queue<int>();

nWorkerThreads = 1000;

nCompletionThreads = 1000;

label2.Text += ", " + nWorkerThreads + " threads";

semaphore = new Semaphore(0, nWorkerThreads);

}

int nWorkerThreads, nCompletionThreads;

Random r;

int GetNum()

{

return r.Next(1, 9);

}

void tr1()

{

while (work)

{

int num = GetNum();

mutex.WaitOne();

queue.Enqueue(num);

int[] arr = new int[queue.Count];

lbBuf.Invoke(new Action(() => lbBuf.Items.Add(num.ToString())));

try

{

semaphore.Release();

}

catch

{

}

mutex.ReleaseMutex();

Thread.Sleep(r.Next(10, 20));

}

}

void tr2(object state)

{

while (work)

{

semaphore.WaitOne();

mutex.WaitOne();

bool has\_elem = queue.Count > 0;

int num = 0;

if (has\_elem)

{

num = queue.Dequeue();

lbBuf.Invoke(new Action(() => lbBuf.Items.RemoveAt(0)));

}

mutex.ReleaseMutex();

string str = "Thread " + Thread.CurrentThread.ManagedThreadId + " work with " + num;

lbWork.Invoke(new Action(() => lbWork.Items.Add(str)));

Thread.Sleep(r.Next(1500, 250000));

lbWork.Invoke(new Action(() => lbWork.Items.Remove(str)));

}

}

Thread t1;

private void btnStart\_Click(object sender, EventArgs e)

{

t1.Start();

for (int i = 0; i < nWorkerThreads; i++)

ThreadPool.QueueUserWorkItem(tr2);

}

private void btnStop\_Click(object sender, EventArgs e)

{

work = false;

Thread.Sleep(300);

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

work = false;

Thread.Sleep(250);

Application.Exit();

}

}

}

**Список используемых источников:**

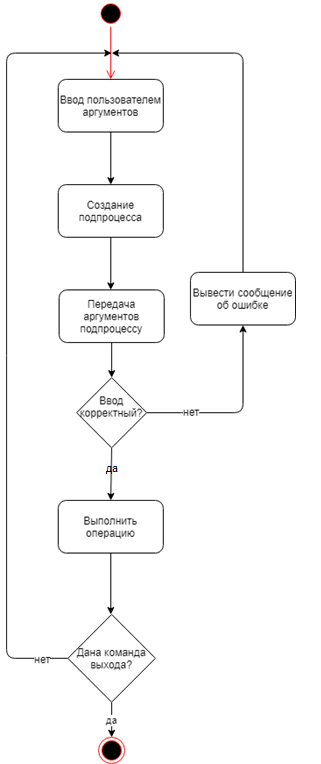
1. <https://metanit.com/sharp/tutorial/11.2.php>
2. <https://habr.com/post/126495/>
3. https://vscode.ru/prog-lessons/potoki-v-si-sharp.htm

**Лабораторная работа № 3**

«Применение вторичных потоков для обслуживания ввода-вывода»

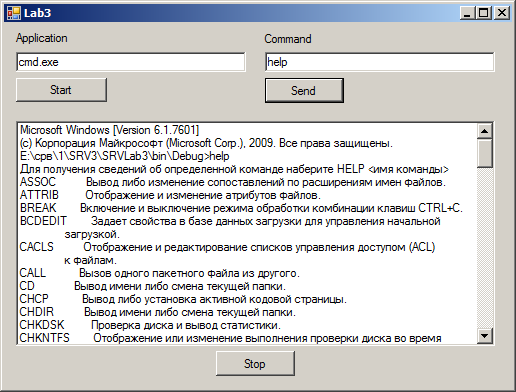
**Задание:** разработать программу которая при вводе корректной информации вызывает консольное приложение, выполняющее определённую задачу.

**Описание выполнения работы:**



Для реализации использовался язык программирования C#, так как он поддерживает параллельное выполнение кода через многопоточность. В C# довольно просто описываются потоки, а также запуск созданных потоков не вызывает трудностей.

Скриншот работы программы:



**Листинг программы:**

namespace SRVLab3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

Process cmd;

private void btnPush\_Click(object sender, EventArgs e)

{

Run();

}

void Send()

{

try

{

var pWriter = cmd.StandardInput;

if (pWriter.BaseStream.CanWrite)

{

pWriter.WriteLine(tbToConsole.Text );

}

else

MessageBox.Show("Cant Write");

pWriter.Flush();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

void Run()

{

if (cmd != null && !cmd.HasExited)

cmd.Kill();

try

{

cmd = new Process

{

StartInfo = new ProcessStartInfo

{

FileName = tbCommand.Text,

RedirectStandardInput = true,

RedirectStandardOutput = true,

UseShellExecute = false,

StandardOutputEncoding = Encoding.GetEncoding(866),

}

};

cmd.OutputDataReceived += new DataReceivedEventHandler(Cmd\_OutputDataReceived);

cmd.Start();

cmd.BeginOutputReadLine();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void Cmd\_OutputDataReceived(object sender, DataReceivedEventArgs e)

{

if (!string.IsNullOrEmpty(e.Data))

lbFromImage.Invoke(new Action(() => lbFromImage.Items.Add(e.Data)));

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!cmd.HasExited)

cmd.Kill();

}

private void btnSend\_Click(object sender, EventArgs e)

{

Send();

}

private void button1\_Click(object sender, EventArgs e)

{

cmd.Kill();

Application.Exit();

}

}

}

**Список используемых источников:**

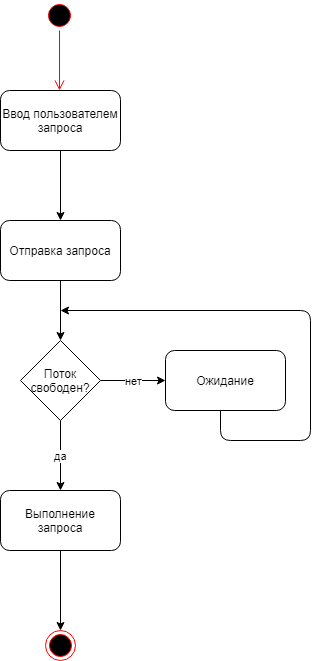
1. <https://metanit.com/sharp/tutorial/11.2.php>
2. <https://habr.com/post/126495/>
3. https://vscode.ru/prog-lessons/potoki-v-si-sharp.htm

**Лабораторная работа № 4**

Задача «Производитель-Потребитель» (Producer-Consumer)

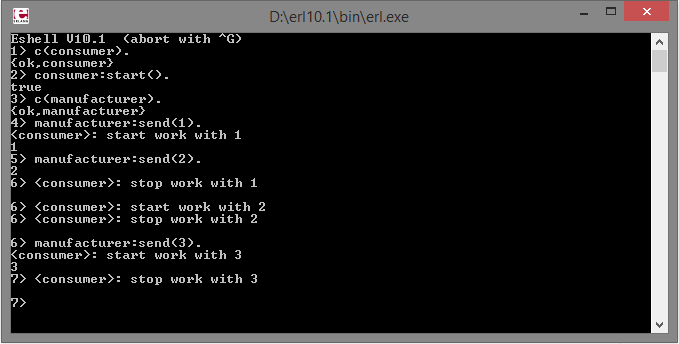
**Задание:** реализовать решение задачи «производитель-потребитель» с применением Erlang.

**Описание выполнения работы:**



Для реализации использовался язык программирования Erlang ввиду его ориентированности на применение потоков.

Скриншот работы программы:



**Листинг программы:**

Consumer.erl:

-module(consumer).

-export([start/0, take/0]).

start() -> Pid = spawn(fun consumer:take/0),

register(atom, Pid).

take() ->

receive

Id ->

io:format("<consumer>: start work with ~p~n" ,[Id]),

timer:sleep(6000),

io:format("<consumer>: stop work with ~p~n~n" ,[Id]),

take()

end.

% Pid = spawn(fun consumer:take/0).

Manufacturer.erl:

-module(manufacturer).

-export([send/1]).

send(Request) ->

atom ! Request.

% receive

% Response ->

% Response

% end.

**Список используемых источников:**

1. <http://www.k-press.ru/cs/2006/3/erlang/erlang.asp>
2. https://www.erlang.org