ГУАП

КАФЕДРА № 44

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | Н. В. Кучин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| ПРОГРАММИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ |
| по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ |
|  |
|  |

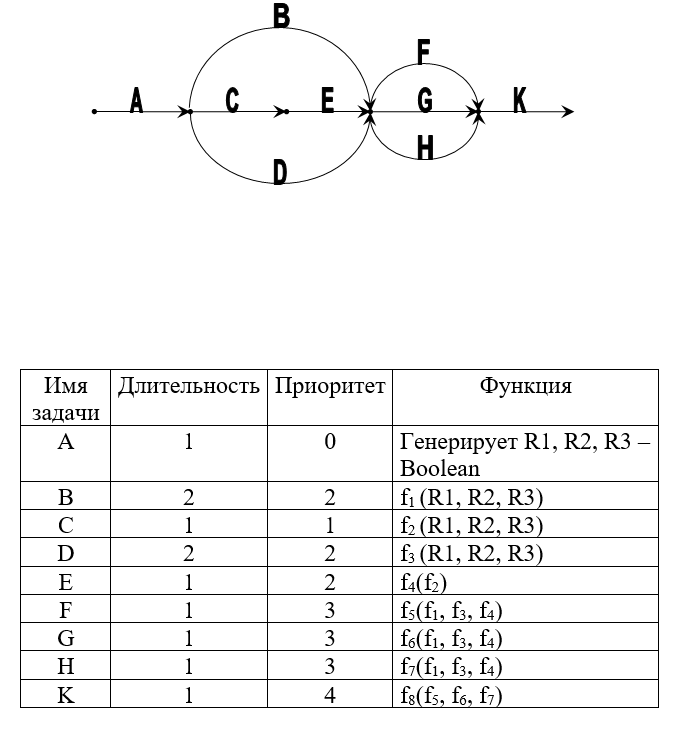
РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4941 |  |  |  | Н.С. Горбунов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**Цель работы:** необходимо написать и отладить программу, которая реализует параллельное выполнение нескольких задач, каждая из которых решает некоторую заданную функцию.

**Вариант 7:**



**Код программы:**

Листинг 1. DataGridModel.cs

using System.ComponentModel;

namespace ThreadProgramming

{

public class DataGridModel : INotifyPropertyChanged // определяет структуру таблицы

{

string timestart;

string taskname;

string result;

string inittaskname;

string timeend;

public string TimeStart

{

get { return timestart; }

set { timestart = value;

OnPropertyChanged("TimeStart");

}

}

public string TaskName

{

get { return taskname; }

set

{

taskname = value;

OnPropertyChanged("TaskName");

}

}

public string Result

{

get { return result; }

set

{

result = value;

OnPropertyChanged("Result");

}

}

public string InitTaskName

{

get { return inittaskname; }

set

{

inittaskname= value;

OnPropertyChanged("InitTaskName");

}

}

public string TimeEnd

{

get { return timeend; }

set

{

timeend = value;

OnPropertyChanged("TimeEnd");

}

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged(string prop = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

}

Листинг 2. VMGrid.cs

using System.Collections.Generic;

using System.ComponentModel;

using System.Collections.ObjectModel;

namespace ThreadProgramming

{

class VMGrid: INotifyPropertyChanged

{

ObservableCollection<DataGridModel> data;

public ObservableCollection<DataGridModel> Data

{

get

{

return data;

}

set

{

data = value;

OnPropertyChanged("Data");

}

}

public VMGrid (List<DataGridModel> data)

{

Data = new ObservableCollection<DataGridModel>(data);

}

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged(string prop = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

}

Листинг 3. TaskWithBackroundWorker.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.ComponentModel;

using System.Windows.Controls;

using System.Windows;

namespace ThreadProgramming

{

//обертка для класса потока

public class TaskWithBackgroundWorker

{

public delegate bool func1(bool x, bool y, bool z); //по заданию может быть два разных типа функций

public delegate bool func2(bool x);

public string name; //имя потока

DateTime StartTime;

public string initName; // имя потока, .который иницализировал

public int delay; // задержка при выполнении функций

public func1 mainF;

public func2 otherF;

public TaskFlag thisFlag;

public int index;

public TaskWithBackgroundWorker task1;

public TaskWithBackgroundWorker task2;

public ProgressBar bar; // элемент GUI

public BackgroundWorker worker; // новый поток

public List<TaskWithBackgroundWorker> StartTasks; // потоки, которе надо будет стартануть

public List<DataGridModel> data;

void DoTask(object sender, DoWorkEventArgs e) // выполнение вычисление. Выполняется в отдельном потоке

{

StartTime = DateTime.Now;

Dictionary<int, TaskFlag> param = (Dictionary<int, TaskFlag>)e.Argument;

initName = param.ContainsKey(255) == true ? param[255].InitName : "A";

for (int i = 0; i <= 100; i++)

{

Thread.Sleep(delay);

(sender as BackgroundWorker).ReportProgress(i);

}

if (param.Count() == 2)

{

e.Result = otherF(param[0].result);

}

else

{

e.Result = mainF(param[0].result, param[1].result, param[2].result);

}

}

void ProgressChanged(object sender, ProgressChangedEventArgs e) // обновление GUI в основном потоке

{

bar.Value = e.ProgressPercentage;

}

void Complete(object sender, RunWorkerCompletedEventArgs e) // выполняется после выполнения DoTask

{

if (StartTasks == null)

{

data.Add(new DataGridModel { TaskName = name, InitTaskName = initName, Result = ((bool)e.Result).ToString(), TimeStart = StartTime.Ticks.ToString(), TimeEnd = DateTime.Now.Ticks.ToString() });

MessageBox.Show("Work Complete! Result: " + (bool)e.Result);

DataWindow window = new DataWindow(data);

window.ShowDialog();

}

else

{

if (task1 != null && task2 != null) //если нет доп потоков, то поток работает в одиночку

{

lock (thisFlag)

{

thisFlag.flag = true;

thisFlag.result = (bool)e.Result;

}

lock (task1.thisFlag)

{

lock (task2.thisFlag)

{

if (task1.thisFlag.flag && task2.thisFlag.flag) // если "соседние" процессы выполнились

{

Dictionary<int, TaskFlag> param = new Dictionary<int, TaskFlag>();

lock (thisFlag)

{

param.Add(index, thisFlag);

param.Add(task1.index, task1.thisFlag);

param.Add(task2.index, task2.thisFlag);

param.Add(255, new TaskFlag { InitName = name });

}

if (!thisFlag.used && !task1.thisFlag.used && !task2.thisFlag.used) //если соседние процессы не просматривались

{

thisFlag.used = true;

task1.thisFlag.used = true;

task2.thisFlag.used = true;

foreach (TaskWithBackgroundWorker i in StartTasks)

{

i.Start(param); //запуск новых потоков

}

}

}

}

}

}

else

{

Dictionary<int, TaskFlag> param = new Dictionary<int, TaskFlag>();

lock (thisFlag)

{

thisFlag.flag = true;

thisFlag.result = (bool)e.Result;

param.Add(index, thisFlag);

param.Add(255, new TaskFlag { InitName = name });

thisFlag.used = true;

}

foreach (TaskWithBackgroundWorker i in StartTasks)

{

i.Start(param);

}

}

}

data.Add(new DataGridModel { TaskName = name, InitTaskName = initName, Result = ((bool)e.Result).ToString(), TimeStart = StartTime.Ticks.ToString(), TimeEnd = DateTime.Now.Ticks.ToString() });

}

public void halfInit(TaskWithBackgroundWorker task1, TaskWithBackgroundWorker task2, List<TaskWithBackgroundWorker> startTasks)

{

this.task1 = task1;

this.task2 = task2;

StartTasks = startTasks;

}

public void Start(Dictionary<int, TaskFlag> param)

{

worker.WorkerReportsProgress = true;

worker.DoWork += DoTask;

worker.ProgressChanged += ProgressChanged;

worker.RunWorkerCompleted += Complete;

worker.RunWorkerAsync(param);

} //запуск потока

}

public class TaskFlag // структура для остлеживание выпонения потоков

{

public bool flag; // флаг выполненности

public bool result; //результат выполнения

public bool used; // проверялся ли флаг

public string InitName; //имя инициализирующего потока

public TaskFlag()

{

flag = false;

used = false;

}

}

}

Листинг 5. DataWindow.xaml

<Window x:Class="ThreadProgramming.DataWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ThreadProgramming"

mc:Ignorable="d"

Title="DataWindow" Height="450" Width="800">

<Grid>

<DataGrid Margin="10" AutoGenerateColumns="True" ItemsSource="{Binding Data}"/>

</Grid>

</Window>

Листинг 6. MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Windows;

using System.ComponentModel;

namespace ThreadProgramming

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

TaskWithBackgroundWorker TaskB; // потоки

TaskWithBackgroundWorker TaskC;

TaskWithBackgroundWorker TaskD;

TaskWithBackgroundWorker TaskE;

TaskWithBackgroundWorker TaskF;

TaskWithBackgroundWorker TaskG;

TaskWithBackgroundWorker TaskH;

TaskWithBackgroundWorker TaskK;

List<DataGridModel> Data;

string initTime;

public MainWindow()

{

InitializeComponent();

Data = new List<DataGridModel>();

}

//функции, исполняемые потоками

public bool f1 (bool x, bool y, bool z)

{

return x && y || (x && z) && (!x && (x || !z));

}

public bool f2(bool x, bool y, bool z)

{

return z && !y || (x && !z) || (!y && !z);

}

public bool f3(bool x, bool y, bool z)

{

return !z || !y && (y && !z) || (!x && z);

}

public bool f4(bool x)

{

return !x;

}

public bool f5(bool x, bool y, bool z)

{

return x && !z && y || (x && z && y);

}

public bool f6(bool x, bool y, bool z)

{

return x && z || (y && z) && (x && y);

}

public bool f7(bool x, bool y, bool z)

{

return (z || y) && (!x && z) && (x || !y);

}

public bool f8(bool x, bool y, bool z)

{

return (x && !y) || (x && !z) || (!y && !z);

}

//обработка кнопки

private void btnDoMagic(object sender, RoutedEventArgs e)

{

DoTaskA();

}

void DoTaskA()

{

//инициализируем оертку потоков

initTime = DateTime.Now.Ticks.ToString();

TaskB = new TaskWithBackgroundWorker { name ="B", delay = 10, data = Data, thisFlag = new TaskFlag(), index = 0, mainF = f1, bar = Bbar, worker = new BackgroundWorker(), task1 = TaskD, task2 = TaskE, StartTasks = new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH } };

TaskC = new TaskWithBackgroundWorker { name = "C", delay = 2, data = Data, thisFlag = new TaskFlag(), index = 0, mainF = f2, bar = Cbar, worker = new BackgroundWorker(), task1 = null, task2 = null, StartTasks = new List<TaskWithBackgroundWorker> { TaskE } };

TaskD = new TaskWithBackgroundWorker { name = "D", delay = 11, data = Data, thisFlag = new TaskFlag(), index = 1, mainF = f3, bar = Dbar, worker = new BackgroundWorker(), task1 = TaskB, task2 = TaskE, StartTasks = new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH } };

TaskE = new TaskWithBackgroundWorker { name = "E", delay = 4, data = Data, thisFlag = new TaskFlag(), index = 2, otherF = f4, bar = Ebar, worker = new BackgroundWorker(), task1 = TaskB, task2 = TaskD, StartTasks = new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH } };

TaskF = new TaskWithBackgroundWorker { name = "F", delay = 7, data = Data, thisFlag = new TaskFlag(), index = 0, mainF = f5, bar = Fbar, worker = new BackgroundWorker(), task1 = TaskG, task2 = TaskH, StartTasks = new List<TaskWithBackgroundWorker> { TaskK } };

TaskG = new TaskWithBackgroundWorker { name = "G", delay = 9, data = Data, thisFlag = new TaskFlag(), index = 1, mainF = f6, bar = Gbar, worker = new BackgroundWorker(), task1 = TaskF, task2 = TaskH, StartTasks = new List<TaskWithBackgroundWorker> { TaskK } };

TaskH = new TaskWithBackgroundWorker { name = "H", delay = 8, data = Data, thisFlag = new TaskFlag(), index = 2, mainF = f7, bar = Hbar, worker = new BackgroundWorker(), task1 = TaskF, task2 = TaskG, StartTasks = new List<TaskWithBackgroundWorker> { TaskK } };

TaskK = new TaskWithBackgroundWorker { name = "K", delay = 10, data = Data, thisFlag = new TaskFlag(), index = 0, mainF = f8, bar = Kbar, worker = new BackgroundWorker(), task1 = null, task2 = null, StartTasks = null };

TaskB.halfInit(TaskD, TaskE, new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH });

TaskC.halfInit(null, null, new List<TaskWithBackgroundWorker> { TaskE });

TaskD.halfInit(TaskB, TaskE, new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH });

TaskE.halfInit(TaskB, TaskD, new List<TaskWithBackgroundWorker> { TaskF, TaskG, TaskH });

TaskF.halfInit(TaskG, TaskH, new List<TaskWithBackgroundWorker> { TaskK });

TaskG.halfInit(TaskF, TaskH, new List<TaskWithBackgroundWorker> { TaskK });

TaskH.halfInit(TaskF, TaskG, new List<TaskWithBackgroundWorker> { TaskK });

//генерация начальных значений

Dictionary<int, TaskFlag> param = new Dictionary<int, TaskFlag>();

param.Add(0, new TaskFlag { result = true });

param.Add(1, new TaskFlag { result = false });

param.Add(2, new TaskFlag { result = true });

Data.Add(new DataGridModel{ InitTaskName = "MAIN", TaskName = "A", Result = "", TimeStart = initTime, TimeEnd = DateTime.Now.Ticks.ToString()});

//запуск потоков

TaskB.Start(param);

TaskC.Start(param);

TaskD.Start(param);

}

}

}

Листинг 7. MainWindow.xaml

<Window x:Class="ThreadProgramming.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ThreadProgramming"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800" ResizeMode="NoResize">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="500"/>

</Grid.ColumnDefinitions>

<StackPanel>

<Image Source="scheme.png" Margin="5"/>

<Button Content="Запуск" Height="25" Width="70" Click="btnDoMagic"/>

</StackPanel>

<StackPanel Grid.Column="1"

x:Name="BarPanel"

Orientation="Vertical">

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task B</Label>

<ProgressBar x:Name="Bbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task C</Label>

<ProgressBar x:Name="Cbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task D</Label>

<ProgressBar x:Name="Dbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task E</Label>

<ProgressBar x:Name="Ebar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task F</Label>

<ProgressBar x:Name="Fbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task G</Label>

<ProgressBar x:Name="Gbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task H</Label>

<ProgressBar x:Name="Hbar" Width="400"/>

</StackPanel>

<StackPanel Margin="5" Orientation="Horizontal">

<Label >Task K</Label>

<ProgressBar x:Name="Kbar" Width="400"/>

</StackPanel>

</StackPanel>

</Grid>

</Window>

Пример работы программы:

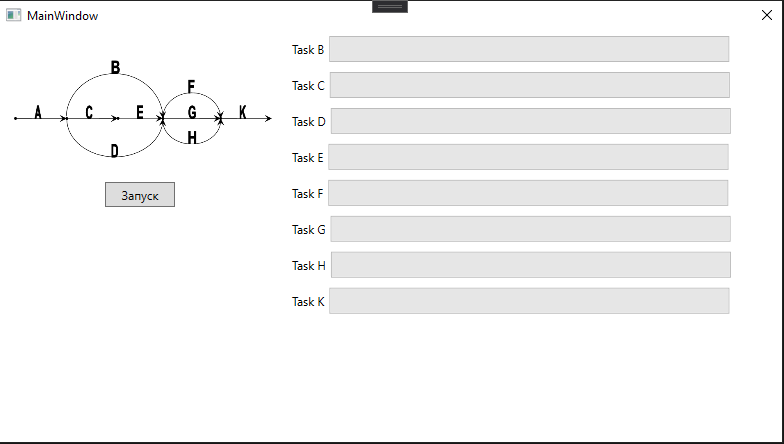


Рисунок 1. Начальное состояние

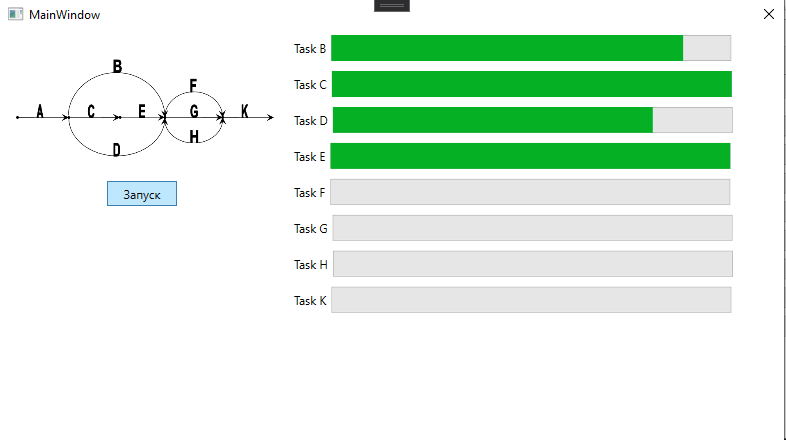


Рисунок 2. Во время выполнения

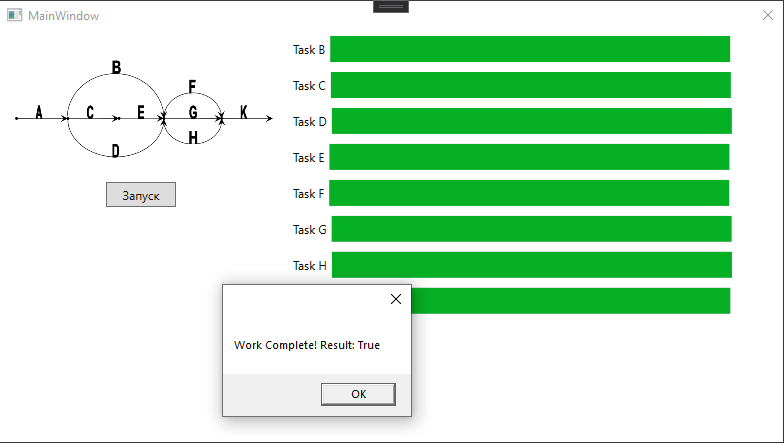


Рисунок 3. Выполнено

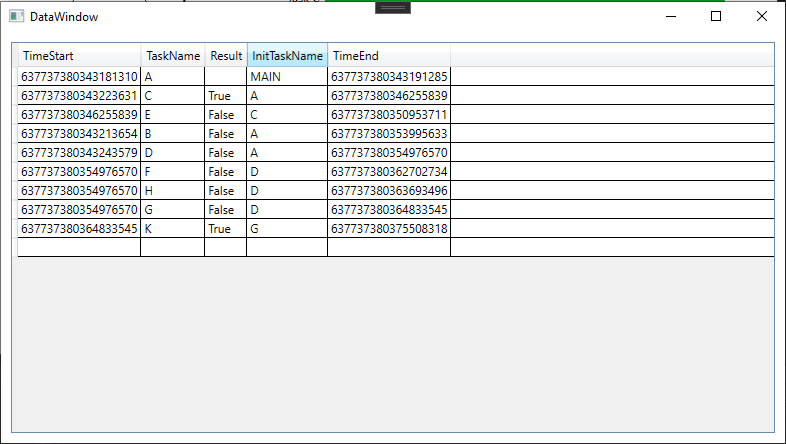


Рисунок 4. Таблица потоков