МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**Отчет по лабораторной работе № 6**

по дисциплине: ”Распределенная и параллельная обработка данных”

на тему: ***”*** ***Планирование асинхронных вычислительных процессов с учетом обмена данных”***

Выполнил**:** студент группы 10701214 Зубарев А. А.

Принял**:** проф. Прихожий А.А.

Минск 2017

# Лабораторная работа 6. Планирование асинхронных вычислительных процессов с учетом обмена данных.

**Цель работы:** изучение стратегии минимизации временной длины плана «Зануление дуг».

## Задание

Спроектировать программу для вычисления назначения задачи на определенный процессор, по заданной таблице плана, количеству операций, времени выполнения операции, матрицы связей операций с другими операциями с указанным временем соединения.

**Теоретическая часть**

Граф задач (task graph) – это ориентированный ациклический взвешенный граф G=(V, E), в котором V – множество узлов, представляющих задачи, E – множество дуг, представляющих передачу данных и отношение предшествования между задачами. Задача определяется как множество инструкций (операторов), выполняемых последовательно на одном процессоре. Соответствующая вершина ni графа метится числом w(ni), описывающим время решения задачи. Дуга (ni, nj) графа метится числом c(i,j), описывающим время передачи данных от задачи ni к задаче nj. Входным называется узел, не имеющий входящих дуг.

Выходным называется узел, не имеющий исходящих дуг. Остальные узлы называются промежуточными. Задача не может начать выполнение не получив данные от родительских задач. Следовательно, условием запуска задачи на выполнение является завершение выполнения всех задач-предшественников на графе задач. При выполнении этого условия задача немедленно запускается на выполнение, следовательно, поведение графа задач в период выполнения является асинхронным. Решение одних задач может происходить параллельно с передачами данных между другими задачами.

Важнейшей характеристикой графа задач является коэффициент «коммуникация/вычисление» (communication-to-computation ratio), определяемый как отношение среднего времени передачи данных от одной задачи к другой к среднему времени решения одной задачи. Время реализации графа задач определяется суммарным весом всех узлов и дуг, входящих в наиболее длинный путь на графе.

**Стратегия планирования «Зануление дуг»**

Стратегия планирования EZ «зануление дуг» стремится сократить длину частично построенного асинхронного плана на каждом шаге планирования путем рассмотрения дуги с максимальным временем передачи данных. Стратегия назначает две задачи, соединенные наиболее «тяжелой» дугой, на один и тот же процессор при условии, что время частичного плана не увеличивается по сравнению с назначением задач на разные процессоры. Если время увеличивается, задачи назначаются на разные подходящие процессоры. Стратегия EZ сначала строит список задач, упорядочивая их в невозрастающем (убывающем) порядке весов, описывающих времена передачи данных. Первая дуга удаляется из списка, а инцидентные узлы-задачи назначаются на один и тот же либо на разные процессоры. Если задачи назначаются на один процессор, дуга зануляется, что интерпретируется как немедленный запуск последующей задачи после завершения предыдущей задачи с нулевым временем обмена данных на одном процессоре. Задачи, назначенные на один процессор, упорядочиваются в соответствии с отношением предшествования и по возрастанию их уровня в графе задач. Процесс планирования заканчивается, когда все задачи назначены на процессоры.

Число шагов работы стратегии меньше числа задач в графе, поскольку рассматриваемое на каждом шаге зануление одной дуги приводит к назначению на процессоры одной или сразу двух задач. Для выбора процессора, на который назначается задача, используется критерий наиболее раннего времени запуска задачи или критерий наиболее короткого во времени частичного плана.

### Код программы

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace L6\_RiPOD

{

public partial class Form1 : Form

{

private EdgeZeroingStrategy EZS;

public Form1()

{

InitializeComponent();

}

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

{

EZS = new EdgeZeroingStrategy();

EZS.File\_Load("start\_param.txt");

EZS.Planning();

ViewResults();

PlanResult();

}

private void ViewResults()

{

ClearView();

// матрица смежности

dataGridView1.ColumnCount = EZS.OperationsCount + 1;

dataGridView1.RowCount = EZS.OperationsCount + 1;

dataGridView1.ColumnHeadersVisible = false;

dataGridView1.RowHeadersVisible = false;

for (int i = 0; i < EZS.OperationsCount + 1; i++)

{

// по вертикали

dataGridView1.Rows[i].Cells[0].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView1.Rows[i].Cells[0].Style.ForeColor = Color.FromArgb(255, 255, 255);

dataGridView1.Rows[i].Cells[0].Value = i;

// по горизонали

dataGridView1.Rows[0].Cells[i].Style.ForeColor = Color.FromArgb(255, 255, 255);

dataGridView1.Rows[0].Cells[i].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView1.Rows[0].Cells[i].Value = i;

dataGridView1.Columns[i].Width = 50;

}

dataGridView1.Rows[0].Cells[0].Value = "Operation \\ operation";

for (int i = 0; i < EZS.OperationsCount; i++)

{

for (int k = 0; k < EZS.TasksConnections[i].Length; k++)

{

if (EZS.TasksConnections[i][k] !="0")

{

dataGridView1.Rows[i + 1].Cells[k + 1].Style.BackColor = Color.FromArgb(82, 97, 160);

}

dataGridView1.Rows[i + 1].Cells[k + 1].Value = EZS.TasksConnections[i][k];

}

}

}

private void ClearView()

{

dataGridView1.ColumnCount = 1;

dataGridView1.RowCount = 1;

dataGridView1.ColumnCount = 1;

dataGridView1.RowCount = 1;

richTextBox1.Text = "";

}

private void PlanResult()

{

int lenPlan = EZS.FindLengthPlan(EZS.ProcessorsList);

richTextBox1.Text = "time";

List<int> power = new List<int>();

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

richTextBox1.Text += "\t\tPE" + i;

power.Add(0);

}

richTextBox1.Text += "\n";

for (int i = 0; i < lenPlan + 1; i++)

{

richTextBox1.Text += i;

for (int pe = 0; pe < EZS.ProcessorsList.Count; pe++)

{

string name = GetTaskNameOnThisTime(EZS.ProcessorsList, i, pe);

richTextBox1.Text += "\t\t" + name;

}

richTextBox1.Text += "\n";

}

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

for (int k = 0; k < EZS.ProcessorsList[i].Count; k++)

{

power[i] += EZS.ProcessorsList[i][k].TimeCalculation;

}

}

richTextBox1.Text += "\nLoad:";

for (int i = 0; i < EZS.ProcessorsList.Count; i++)

{

double load = Convert.ToDouble(power[i]) / Convert.ToDouble(lenPlan);

richTextBox1.Text += "\t\t" + load.ToString(String.Format("##.## %"));

}

}

private string GetTaskNameOnThisTime(List<List<EZTask>> procList, int time, int pe)

{

foreach (var task in procList[pe])

{

if (time >= task.StartTime && time <= task.EndTime)

{

return task.Name;

}

}

return " ";

}

}

}

**EZTask.cs**

using System.Collections.Generic;

namespace L6\_RiPOD

{

struct EZTask

{

public string Name { get; set; }

public int TimeCalculation { get; set; }

public int StartTime { get; set; }

public int EndTime { get; set; }

public List<Connection> Connections;

}

}

**Connection.cs**

namespace L6\_RiPOD

{

struct Connection

{

public string BeginTaskName { get; set; }

public string TaskName { get; set; }

public int TimeConnection { get; set; }

public static bool operator ==(Connection con1, Connection con2)

{

if (con1.BeginTaskName == con2.BeginTaskName &&

con1.TaskName == con2.TaskName &&

con1.TimeConnection == con2.TimeConnection)

return true;

return false;

}

public static bool operator !=(Connection con1, Connection con2)

{

if (con1.BeginTaskName == con2.BeginTaskName &&

con1.TaskName == con2.TaskName &&

con1.TimeConnection == con2.TimeConnection)

return false;

return true;

}

}

}

**EdgeZeroingStrategy.cs**

using System;

using System.Collections.Generic;

using System.IO;

namespace L6\_RiPOD

{

class EdgeZeroingStrategy

{

private int[] ArrayTimeCalculation;

// для загрузки из файла

private List<List<List<int>>> conList = new List<List<List<int>>>();

// список соединений, которые еще не использованы

private List<Connection> ConnectList = new List<Connection>();

public int OperationsCount { get; set; }

public string[][] TasksConnections;

public List<EZTask> Tasks = new List<EZTask>();

public List<List<EZTask>> ProcessorsList = new List<List<EZTask>>();

// конструктор

public EdgeZeroingStrategy()

{

OperationsCount = 0;

}

#region Чтение данных из файла

// чтение начальных данных из файла

public void File\_Load(string filestr)

{

// получение данных из файла

StreamReader file = new StreamReader(filestr);

//step 1 чтение кол-ва операций

string buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

OperationsCount = Convert.ToInt32(buff);

//step 2 чтение времени выполнения каждой задачи

buff = file.ReadLine();

buff = Find\_value(buff);

string[] arr = buff.Split(' ');

ArrayTimeCalculation = new int[arr.Length];

for (var i = 0; i < arr.Length; i++)

{

ArrayTimeCalculation[i] = Convert.ToInt32(arr[i]);

}

//step 6 чтение таблицы связей

file.ReadLine();

TasksConnections = new string[OperationsCount][];

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

{

buff = file.ReadLine();

List<List<int>> list = new List<List<int>>();

list.Add(new List<int>());

list.Add(new List<int>());

string[] arrInts = buff.Split(' ');

TasksConnections[i] = arrInts;

for (var k = 0; k < arrInts.Length; k++)

{

var ai = arrInts[k];

if (ai != "0")

{

list[0].Add(k);

list[1].Add(Int32.Parse(ai));

}

}

conList.Add(list);

}

// сборка всех полученных данных в список структур типа ezTask

CollectTasks();

// Закрытие файла

file.Close();

}

// поиск строки после символа-разделителя

private string Find\_value(string s)

{

int index = s.IndexOf(':') + 2;

return s.Substring(index); // извлечение подстроки с указанной позиции и до конца

}

// сборка полученных данных из файла в 1 коллекцию

private void CollectTasks()

{

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

{

EZTask ezTask = new EZTask();

ezTask.Name = "n" + (i + 1);

ezTask.TimeCalculation = ArrayTimeCalculation[i];

ezTask.StartTime = 0;

ezTask.Connections = new List<Connection>();

for (int k = 0; k < conList[i][0].Count; k++)

{

Connection connection = new Connection();

int a = conList[i][0][k] + 1;

connection.TaskName = "n" + a;

connection.BeginTaskName = ezTask.Name;

connection.TimeConnection = conList[i][1][k];

ezTask.Connections.Add(connection);

}

Tasks.Add(ezTask);

}

}

#endregion

public void Planning()

{

SortTasks();

EZ();

}

#region Сортировка задач

private void SortTasks()

{

List<EZTask> ezTasks = new List<EZTask>();

foreach (var task in Tasks)

{

EZTask t = new EZTask();

t = task;

List<Connection> connections = CopyConnectionsList(task.Connections);

connections = SortConnectionsList(connections); // сортировка соединений 1 задачи

t.Connections = connections;

ezTasks.Add(t);

}

Tasks = ezTasks;

GetConnectList();

ConnectList = SortConnectionsList(ConnectList);

}

// сортировка соединений 1 задачи

private List<Connection> SortConnectionsList(List<Connection> list)

{

list.Sort((connection, connection1) =>

{

if (connection.TimeConnection < connection1.TimeConnection)

{

return 1;

}

if (connection.TimeConnection > connection1.TimeConnection)

{

return -1;

}

return 0;

});

return list;

}

private void GetConnectList()

{

for (int i = 0; i < Tasks.Count; i++)

{

for (int k = 0; k < Tasks[i].Connections.Count; k++)

{

Connection c = new Connection();

c.TimeConnection = Tasks[i].Connections[k].TimeConnection;

c.BeginTaskName = Tasks[i].Connections[k].BeginTaskName;

c.TaskName = Tasks[i].Connections[k].TaskName;

ConnectList.Add(c);

}

}

}

#endregion

#region Копирование задачи и различных списков

// Глубокое копирование списка соединений

private List<Connection> CopyConnectionsList(List<Connection> list)

{

List<Connection> newList = new List<Connection>();

foreach (var inList in list)

{

Connection con = new Connection();

con.TimeConnection = inList.TimeConnection;

con.BeginTaskName = inList.BeginTaskName;

con.TaskName = inList.TaskName;

newList.Add(con);

}

return newList;

}

// копирование задачи

private EZTask CopyTask(EZTask task)

{

EZTask newTask = new EZTask();

newTask.Name = task.Name;

newTask.StartTime = task.StartTime;

newTask.EndTime = task.EndTime;

newTask.TimeCalculation = task.TimeCalculation;

newTask.Connections = CopyConnectionsList(task.Connections);

return newTask;

}

// копирование плана

private List<List<EZTask>> CopyProcessorList(List<List<EZTask>> procList)

{

List<List<EZTask>> list = new List<List<EZTask>>();

for (int i = 0; i < procList.Count; i++)

{

List<EZTask> ezTasks = new List<EZTask>();

for (int k = 0; k < procList[i].Count; k++)

{

ezTasks.Add(CopyTask(procList[i][k]));

}

list.Add(ezTasks);

}

return list;

}

#endregion

#region План "Зануление дуг"

#region Планирование

private void EZ()

{

while (ConnectList.Count > 0)

{

SetTaskInPlane(ProcessorsList, ConnectList[0]);

// удаление дуги из сортированного списка

ConnectList.RemoveAt(0);

}

}

// Установление задачи в план

private void SetTaskInPlane(List<List<EZTask>> procList, Connection connection)

{

// если нет в плане 2-х задач из connection, то добавить в новый процессор

if (PlaneNotContainsCoupleTasks(procList, connection))

{

return;

}

// если есть в плане 2-х задач из connection, то каскадно изменить точки начала и конца

if (PlaneContainsCoupleTasks(procList, connection))

{

return;

}

// если есть в плане первая задача из connection

if (PlaneContainsOneTask(procList, connection))

{

return;

}

}

// если нет в плане 2-х задач из connection, то добавить в новый процессор

private bool PlaneNotContainsCoupleTasks(List<List<EZTask>> procList, Connection connection)

{

if (!ProcListContains(procList, connection.BeginTaskName) &

!ProcListContains(procList, connection.TaskName))

{

List<EZTask> list = new List<EZTask>();

// поиск индекса задачи в глобальном массиве задач

int indexTask = FindTask(connection.BeginTaskName);

// копирование данных, так как имеются ссылочные типы

EZTask task = CopyTask(Tasks[indexTask]);

// установление конечного времени для 1 задачи

task.EndTime = task.TimeCalculation;

task.Connections.Clear();

Connection connect = new Connection();

connect.TimeConnection = 0; // зануление дуги

connect.BeginTaskName = connection.BeginTaskName;

connect.TaskName = connection.TaskName;

task.Connections.Add(connect);

// добавление в список

list.Add(task);

// поиск индекса задачи в глобальном массиве задач

indexTask = FindTask(connection.TaskName);

// копирование данных, так как имеются ссылочные типы

EZTask task2 = CopyTask(Tasks[indexTask]);

// установление конечного времени для 2 задачи

task2.StartTime = task.EndTime;

task2.EndTime = task2.TimeCalculation + task2.StartTime;

task2.Connections.Clear();

// добавление в список

list.Add(task2);

// создание нового процессора и добавление 2-х задач

procList.Add(list);

return true;

}

return false;

}

// если есть в плане 2 задачи из connection

private bool PlaneContainsCoupleTasks(List<List<EZTask>> procList, Connection connection)

{

if (ProcListContains(procList, connection.BeginTaskName) &

ProcListContains(procList, connection.TaskName))

{

foreach (var proc in procList)

{

foreach (var p in proc)

{

if (p.Name == connection.BeginTaskName)

{

Connection newConnection = new Connection();

if (FindIdProcessor(procList, connection.BeginTaskName) ==

FindIdProcessor(procList, connection.TaskName))

{

newConnection.TimeConnection = 0; // зануление дуги

}

else

{

newConnection.TimeConnection = connection.TimeConnection;

}

newConnection.TaskName = connection.TaskName;

newConnection.BeginTaskName = connection.BeginTaskName;

// добавленеи новой связи к 1 задаче

p.Connections.Add(newConnection);

// каскадное смещение времени зависимых задач

CascadeChangeTime(procList, connection.BeginTaskName);

break;

}

}

}

return true;

}

return false;

}

// если есть в плане 1 задача из connection

private bool PlaneContainsOneTask(List<List<EZTask>> procList, Connection connection)

{

bool firstTask;

int idProcessor;

if (ProcListContains(procList, connection.BeginTaskName) &

!ProcListContains(procList, connection.TaskName))

{

firstTask = false;

idProcessor = FindIdProcessor(procList, connection.BeginTaskName);

}

else

{

if (!ProcListContains(procList, connection.BeginTaskName) &

ProcListContains(procList, connection.TaskName))

{

firstTask = true;

idProcessor = FindIdProcessor(procList, connection.TaskName);

}

else

{

return false;

}

}

// список времени длины плана разных вариантов

List<int> lengthPlanList = new List<int>();

// если в списке времени длины будут неск. одинаковых минимальных длин

// и среди них есть индекс, то назначить по этому индексу

int indexPriority = -1;

// проверка вариантов назначения на каждый процессор

for (int i = 0; i < procList.Count + 1; i++)

{

EZTask insertFirstTask = CopyTask(Tasks[FindTask(connection.BeginTaskName)]);

EZTask insertSecondTask = CopyTask(Tasks[FindTask(connection.TaskName)]);

insertFirstTask.Connections.Clear();

insertSecondTask.Connections.Clear();

// копирование списка для сохранения оригинала

List<List<EZTask>> newProcList = CopyProcessorList(procList);

if (i == idProcessor)

{

indexPriority = i;

if (firstTask)

{

InsertFirstTaskInPlane(newProcList, i, true,

insertFirstTask, connection);

}

else

{

InsertSecondTaskInPlane(newProcList, i, true,

insertSecondTask, connection);

}

}

else

{

if (firstTask)

{

InsertFirstTaskInPlane(newProcList, i, false,

insertFirstTask, connection);

}

else

{

InsertSecondTaskInPlane(newProcList, i, false,

insertSecondTask, connection);

}

}

// поиск длины плана и занесение в список

lengthPlanList.Add(FindLengthPlan(newProcList));

}

// поиск минимального

int minValue = Int32.MaxValue;

for (int i = 0; i < lengthPlanList.Count; i++)

{

if (minValue > lengthPlanList[i])

{

minValue = lengthPlanList[i];

}

}

List<int> minList = new List<int>();

for (int i = 0; i < lengthPlanList.Count; i++)

{

if (minValue == lengthPlanList[i])

{

minList.Add(i);

}

}

// Главный вариант плана

int variantProc = -1;

if (minList.Count > 1)

{

if (minList.Contains(indexPriority))

{

variantProc = indexPriority;

}

else

{

variantProc = minList[0];

}

}

else

{

variantProc = minList[0];

}

EZTask insertFirstTaskInPlane = CopyTask(Tasks[FindTask(connection.BeginTaskName)]);

EZTask insertSecondTaskInPlane = CopyTask(Tasks[FindTask(connection.TaskName)]);

insertFirstTaskInPlane.Connections.Clear();

insertSecondTaskInPlane.Connections.Clear();

// вставка элемента по заданному id процессора

if (variantProc == indexPriority)

{

if (firstTask)

{

InsertFirstTaskInPlane(procList, variantProc, true,

insertFirstTaskInPlane, connection);

}

else

{

InsertSecondTaskInPlane(procList, variantProc, true,

insertSecondTaskInPlane, connection);

}

}

else

{

if (firstTask)

{

InsertFirstTaskInPlane(procList, variantProc, false,

insertFirstTaskInPlane, connection);

}

else

{

InsertSecondTaskInPlane(procList, variantProc, false,

insertSecondTaskInPlane, connection);

}

}

return true;

}

#endregion

#region Вставка задачи

// вставка второй задачи в план на определенный процессор

private void InsertSecondTaskInPlane(List<List<EZTask>> procList, int procId,

bool equalsIdProc, EZTask task, Connection connection)

{

// если назначение на новый процессор

if (procId > procList.Count - 1)

{

List<EZTask> newProc = new List<EZTask>();

task.StartTime = 0;

task.EndTime = task.TimeCalculation;

newProc.Add(task);

procList.Add(newProc);

}

else

{

EZTask lastTask = procList[procId][procList[procId].Count - 1];

if (equalsIdProc)

{

task.StartTime = lastTask.EndTime;

}

else

{

task.StartTime = lastTask.EndTime + connection.TimeConnection;

// мнимая связь

Connection con = new Connection();

con.BeginTaskName = lastTask.Name;

con.TaskName = connection.TaskName;

con.TimeConnection = 0;

lastTask.Connections.Add(con);

// сохранение добавления мнимой связи

procList = ReplaceTaskInProcList(procList, lastTask.Name, lastTask);

}

task.EndTime = task.StartTime + task.TimeCalculation;

procList[procId].Add(task);

}

int PE, i;

// нахождение 1 задачи из соединения и добавление нового соединения

EZTask firstTask = FindTaskInProcList(procList, connection.BeginTaskName, out PE, out i);

Connection newConnection = new Connection();

newConnection.BeginTaskName = connection.BeginTaskName;

newConnection.TaskName = connection.TaskName;

if (equalsIdProc)

{

newConnection.TimeConnection = 0;

}

else

{

newConnection.TimeConnection = connection.TimeConnection;

}

firstTask.Connections.Add(newConnection);

procList = ReplaceTaskInProcList(procList, firstTask.Name, firstTask);

// каскадное изменение времени

CascadeChangeTime(procList, firstTask.Name);

}

// вставка первой задачи в план на определенный процессор

private void InsertFirstTaskInPlane(List<List<EZTask>> procList, int procId,

bool equalsIdProc, EZTask task, Connection connection)

{

// если назначение на новый процессор

if (procId > procList.Count - 1)

{

List<EZTask> newProc = new List<EZTask>();

task.StartTime = 0;

task.EndTime = task.TimeCalculation;

newProc.Add(task);

procList.Add(newProc);

}

else

{

EZTask lastTask = procList[procId][procList[procId].Count - 1];

Connection con = new Connection();

con.BeginTaskName = connection.BeginTaskName;

con.TaskName = connection.TaskName;

if (equalsIdProc)

{

int PE, i;

EZTask secondTask = FindTaskInProcList(procList, connection.TaskName, out PE, out i);

task.StartTime = secondTask.StartTime;

con.TimeConnection = 0;

procList = ChangePreviousTaskConnection(procList, connection.TaskName, connection.BeginTaskName);

}

else

{

task.StartTime = lastTask.EndTime;

con.TimeConnection = connection.TimeConnection;

}

task.Connections.Add(con);

task.EndTime = task.StartTime + task.TimeCalculation;

procList[procId].Add(task);

}

// каскадное изменение времени

CascadeChangeTime(procList, connection.BeginTaskName);

}

#endregion

#region Вспомогательные методы

// проверка на наличие задачи в плане

private bool ProcListContains(List<List<EZTask>> procList, string taskName)

{

foreach (var proc in procList)

{

foreach (var p in proc)

{

if (p.Name == taskName)

return true;

}

}

return false;

}

// каскадное изменение времени выполнения

private int ErrorRecurcy = 0; // число безопасности зацикливания рекурсии

private void CascadeChangeTime(List<List<EZTask>> procList, string taskName)

{

ErrorRecurcy++;

if (ErrorRecurcy > 1000)

{

throw new Exception();

}

if (taskName == null)

{

return;

}

int PE, i;

EZTask task = FindTaskInProcList(procList, taskName, out PE, out i);

foreach (var connection in task.Connections)

{

EZTask nextTask = FindTaskInProcList(procList, connection.TaskName, out PE, out i);

if (nextTask.EndTime != 0)

{

if (nextTask.StartTime < task.EndTime + connection.TimeConnection)

{

nextTask.StartTime = task.EndTime + connection.TimeConnection;

nextTask.EndTime = nextTask.StartTime + nextTask.TimeCalculation;

procList = ReplaceTaskInProcList(procList, nextTask.Name, nextTask);

}

}

CascadeChangeTime(procList, nextTask.Name);

}

}

// замена задачи в плане

private List<List<EZTask>> ReplaceTaskInProcList(List<List<EZTask>> procList,

string taskName, EZTask replaceTask)

{

int PE, i;

FindTaskInProcList(procList, taskName, out PE, out i);

procList[PE].RemoveAt(i);

procList[PE].Insert(i, replaceTask);

return procList;

}

// перенаправление всех связей, установленных к taskName, в newTaskName

private List<List<EZTask>> ChangePreviousTaskConnection(List<List<EZTask>> procList,

string taskName, string newTaskName)

{

for (int i = 0; i < procList.Count; i++)

{

for (int k = 0; k < procList[i].Count; k++)

{

for (int t = 0; t < procList[i][k].Connections.Count; t++)

{

if (procList[i][k].Connections[t].TaskName == taskName)

{

Connection con = new Connection();

con.BeginTaskName = procList[i][k].Connections[t].BeginTaskName;

con.TimeConnection = procList[i][k].Connections[t].TimeConnection;

con.TaskName = newTaskName;

procList[i][k].Connections.RemoveAt(t);

procList[i][k].Connections.Insert(t, con);

}

}

}

}

return procList;

}

#region Функции поиска

// поиск индекса задачи в глобальном массиве задач

private int FindTask(string name)

{

for (int i = 0; i < Tasks.Count; i++)

{

if (Tasks[i].Name == name)

return i;

}

return -1;

}

// Поиск номера процессора, где находится задача

private int FindIdProcessor(List<List<EZTask>> procList, string taskName)

{

for (int i = 0; i < procList.Count; i++)

{

foreach (var task in procList[i])

{

if (task.Name == taskName)

{

return i;

}

}

}

return -1;

}

// поиск задачи по имени в списке задач и возвращение ссылки на нее

private EZTask FindTaskInProcList(List<List<EZTask>> procList, string taskName,

out int PE, out int index)

{

for (int i = 0; i < procList.Count; i++)

{

for (int k = 0; k < procList[i].Count; k++)

{

if (procList[i][k].Name == taskName)

{

PE = i;

index = k;

return procList[i][k];

}

}

}

EZTask nullTask = new EZTask();

nullTask.EndTime = 0;

PE = -1;

index = -1;

return nullTask;

}

// поиск длины плана и на каком процессоре макс. длина

public int FindLengthPlan(List<List<EZTask>> procList)

{

int len = 0;

for (int i = 0; i < procList.Count; i++)

{

foreach (var task in procList[i])

{

if (len < task.EndTime)

{

len = task.EndTime;

}

}

}

return len;

}

#endregion

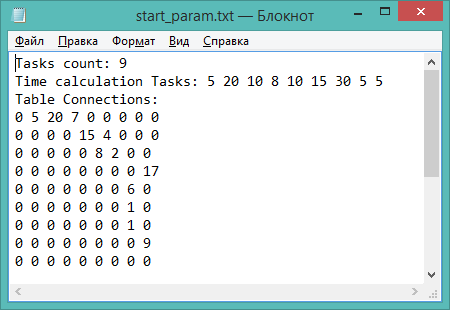
#endregion

#endregion

}

}

### Скриншоты результатов



## Вывод

Изучен стратегию минимизации времени длины асинхронного плана.