Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рыбинский государственный авиационный технический университет

имени П.А. Соловьева»

Факультет радиоэлектроники и информатики

Кафедра Математическое и программное обеспечение ЭВС

**ОТЧЕТ**

**по лабораторной работе №7**

**R**EADER 1

Студент группы ИПБ-15 Козин О.И .

*(Код) (Подпись, дата) (Фамилия И. О.)*

Рыбинск 2019

Оглавление

[Цель 2](#_Toc5026380)

[Простейшие взаимодействия 2](#_Toc5026381)

[Архитектура 2](#_Toc5026382)

[Анализ работы программ 3](#_Toc5026383)

[Управляющие автоматы для задачи 4](#_Toc5026384)

[Дискретная система программы на языке c# 6](#_Toc5026385)

[Приложение. Коды планировщиков. 7](#_Toc5026386)

[Приложение. Интерпретация временных диаграмм 16](#_Toc5026387)

# Цель

1. Разработать автоматную модель задачи обедающие философы
2. Научиться переводить автоматную модель в управляющую программу ADA
3. Запрограммировать решение задачи в соответствии с автоматной схемой.

# Простейшие взаимодействия

## Архитектура

Table

Check\_neighbor

Check\_place

Client

Servant

Client

Philosopher

Want to eat

3

1

2

Finish\_eat

4

На рис.1 представлена архитектура решения задачи.

## Анализ работы программ

Ниже представлен фрагмент временных диаграмм работы программы с соответственно автоматным схемам.

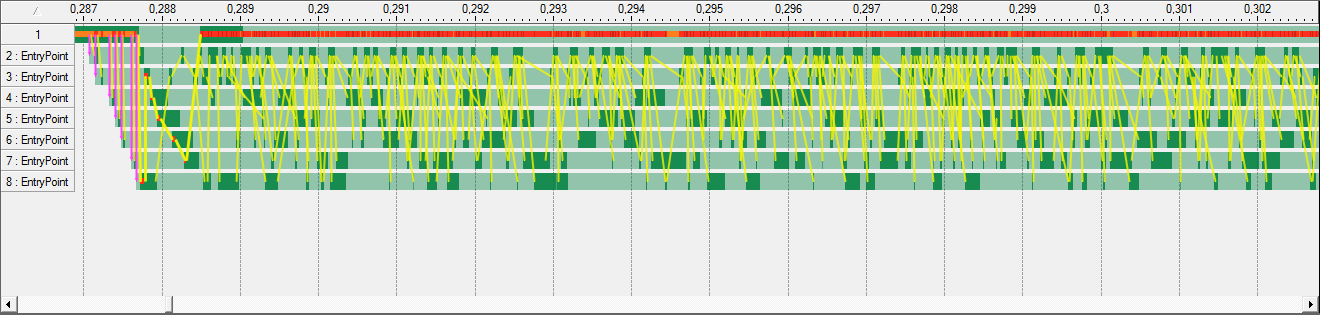


Рис.2 временная диаграмма работы программы.

## Управляющие автоматы для задачи

call

founded

No place

Рис. 3. Автоматная схема Слуги

Find place

time

Can’t eat

time

Can eat

done

Рис. 4. Автоматная схема Философов

Check place

Finish eat

Check neighbor

Рис. 5. Автоматная схема Стола

## Дискретная система программы на языке c#

Филосов

Слуга

Y1

Y2

Y3

X3

X1

X2

Y1 = δ1 (want to eat, x3, филосов)

Y2 = δ2 (free res, x3, филосов)

Y3 = δ3 (take place, x1, слуга)

# Приложение. Коды планировщиков.

Вариант задачи на ADA.

--with Gnat.IO;

with Text\_Io;

with Random\_Generic;

procedure p01 is

--use Ada.Text\_IO;

use Text\_Io;

subtype my\_type is integer range 0..2;

package rnd\_choise is

new Random\_Generic (Result\_Subtype => my\_type);

subtype my\_type2 is integer range 1000..20000;

package rnd\_choise2 is

new Random\_Generic (Result\_Subtype => my\_type2);

---------------------------------------------------------------------

task table is

entry check\_place(data : out integer);

entry finish\_eat(data : in integer);

entry check\_neighbours (data : in out integer);

end table;

task type Phil (lacks : integer) is

end Phil;

task servant is

entry want\_to\_eat(data : in out integer);

end servant;

-----------------------------------------------------------------------------------

task body table is

--L: my\_type;

sits: array (1..5) of integer;

forks: array (1..10) of integer;

blocks: array (1..5) of integer;

cnt:integer;

left, right, cur :integer;

begin

cnt:=0;

for I in sits'Range loop

sits(I):=0;

blocks(I):=0;

forks(I):=0;

forks(I\*2):=0;

end loop;

loop

select

accept finish\_eat(data:in integer) do

--put\_line(" (table) cnt = " & integer'Image(cnt) & " ================================-1");

--put\_line(" [" & integer'Image(sits(1)) & " , " & integer'Image(sits(2)) & " , " & integer'Image(sits(3)) & " , " & integer'Image(sits(4)) & " , " & integer'Image(sits(5)) & "]");

sits (data):=0;

forks (data\*2):=0;

forks (data\*2 -1):=0;

--put\_line(" phil leave" & integer'Image(data) & " place");

--put\_line(" [" & integer'Image(sits(1)) & " , " & integer'Image(sits(2)) & " , " & integer'Image(sits(3)) & " , " & integer'Image(sits(4)) & " , " & integer'Image(sits(5)) & "]");

cnt:=cnt-1;

--put\_line(" (table) cnt = " & integer'Image(cnt) & " ================================-2");

end finish\_eat;

or

accept check\_neighbours (data : in out integer) do

--put\_line(" [" & integer'Image(sits(1)) & " , " & integer'Image(sits(2)) & " , " & integer'Image(sits(3)) & " , " & integer'Image(sits(4)) & " , " & integer'Image(sits(5)) & "]");

cur:=data;

sits(cur):=3;

left:=(cur-1);

if (left = 0)

then

left:=5;

end if;

right:=(cur+1) mod 5;

--Put\_Line(" (table) left = " & integer'Image(left) & " ;cur = " & integer'Image(cur) & " ;right = " & integer'Image(right));

data:=sits(left) + sits(right);

if data > 2

then

sits(cur):=1;

end if;

--put\_line(" [" & integer'Image(sits(1)) & " , " & integer'Image(sits(2)) & " , " & integer'Image(sits(3)) & " , " & integer'Image(sits(4)) & " , " & integer'Image(sits(5)) & "]");

end check\_neighbours;

or

accept check\_place (data : out integer) do

data:=0;

--put\_line(" (table) cnt = " & integer'Image(cnt) & " ================================1");

if (cnt < 4)

then

data:=0;

for I in sits'Range loop

if (I mod 2) /= 0

then

if (sits (I) = 0)

then

data:= I;

sits (I):=1;

exit;

end if;

end if;

end loop;

if data = 0

then

for I in sits'Range loop

if (I mod 2) = 0

then

if (sits (I) = 0)

then

data:= I;

sits (I):=1;

exit;

end if;

end if;

end loop;

end if;

else

--put\_line(" (table) cnt = " & integer'Image(cnt) & " ================================2");

data:=0;

end if;

if (data /= 0)

then

cnt:=cnt+1;

--put\_line(" (table) cnt++");

end if;

--put\_line(" (table) cnt = " & integer'Image(cnt) & " ================================2");

end check\_place;

end select;

end loop;

end table;

------------------------------------------------------------------------------------------------

task body servant is

-- L:my\_type;

place\_l:integer;

begin

loop

accept want\_to\_eat(data : in out integer) do

--put\_line(" (servant) serving phil " & integer'Image(data));

loop

--put\_line(" (servant) atempt to serve phil" & integer'Image(data));

table.check\_place(place\_l);

--put\_line(" (servant) result of phil " & integer'Image(data) & " is " & integer'Image(place\_l));

exit when place\_l /= 0;

--put\_line(" (servant) Table is full!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

end loop;

data:= place\_l;

end want\_to\_eat;

end loop;

end servant;

------------------------------------------------------------------------------------------

task body Phil is

-- L: my\_type;

a: integer;

n: my\_type2;

my\_place, data: integer;

begin

my\_place:=lacks;

loop

--delay 0.0001;

n:= rnd\_choise2.random\_value;

--Put\_Line("!!" & Integer'Image(n));

for I in 1..n loop

a := i\*i-1;

end loop;

--put\_line("(philosopher) #" & integer'Image(lacks) & " wana eat");

my\_place:=lacks;

servant.want\_to\_eat(my\_place);

--put\_line("(philosopher) #" & integer'Image(lacks) & " take place #"& integer'Image(my\_place) );

loop

data:=my\_place;

--put\_line("(philosopher) #" & integer'Image(lacks) & " atempt to eat");

table.check\_neighbours(data);

exit when data <3;

--put\_line("(philosopher) #" & integer'Image(lacks) & " failse to eat(((((");

end loop;

--put\_line("(philosopher) #" & integer'Image(lacks) & " eating!!!!!!!!!!");

n:= rnd\_choise2.random\_value;

--Put\_Line("!!" & Integer'Image(n));

for I in 1..n loop

a := i\*i-1;

end loop;

--delay 0.0001;

--put\_line("(philosopher) #" & integer'Image(lacks) & " wana leave");

table.finish\_eat(my\_place);

end loop;

end Phil;

T\_1 : Phil (lacks => 1);

T\_2 : Phil (lacks => 2);

T\_3 : Phil (lacks => 3);

T\_4 : Phil (lacks => 4);

T\_5 : Phil (lacks => 5);

-------------------------------------------------------

begin

Put\_Line("Hello, world!");

end p01;

Решение на c#

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

using System.Diagnostics;

namespace DiningPhilosophers

{

class Program

{

private const int PHILOSOPHER\_COUNT = 5;

static void Main(string[] args)

{

Servant serv = new Servant();

var philosophers = InitializePhilosophers(serv);

Console.WriteLine("Dinner is starting!"); // Start

var philosopherThreads = new List<Thread>();

foreach (var philosopher in philosophers)

{

var philosopherThread = new Thread(new ThreadStart(philosopher.EatAll));

philosopherThreads.Add(philosopherThread);

philosopherThread.Start();

}

foreach (var thread in philosopherThreads) // finish eating

{

thread.Join();

}

Console.WriteLine("Dinner is over!");

Console.ReadKey();

}

private static List<Philosopher> InitializePhilosophers(Servant serv)

{

var philosophers = new List<Philosopher>(PHILOSOPHER\_COUNT);

for (int i = 0; i < PHILOSOPHER\_COUNT; i++)

{

philosophers.Add(new Philosopher(philosophers, i, serv));

}

foreach (var philosopher in philosophers) // Assign chopsticks to philosophers

{

philosopher.LeftChopstick = philosopher.LeftPhilosopher.RightChopstick ?? new Chopstick();

philosopher.RightChopstick = philosopher.RightPhilosopher.LeftChopstick ?? new Chopstick();

}

return philosophers;

}

}

public class Servant

{

public int x = 0;

object locker = new object();

public bool takePlace()

{

lock (locker)

Console.WriteLine(x);

{

if (x < 4)

{

x++;

return true;

}

else

{

return false;

}

}

}

public void freePlace()

{

x--;

}

}

[DebuggerDisplay("Name = {Name}")]

public class Philosopher

{

private const int TIMES\_TO\_EAT = 5;

private int \_timesEaten = 0;

private readonly List<Philosopher> \_allPhilosophers;

private readonly int \_index;

private Servant \_serv;

private object locker = new object();

public static Random rnd = new Random();

public Philosopher(List<Philosopher> allPhilosophers, int index, Servant serv)

{

\_serv = serv;

\_allPhilosophers = allPhilosophers;

\_index = index;

this.Name = string.Format("Philosopher {0}", \_index);

this.State = State.Thinking;

}

public string Name { get; private set; }

public State State { get; private set; }

public Chopstick LeftChopstick { get; set; }

public Chopstick RightChopstick { get; set; }

public Philosopher LeftPhilosopher

{

get

{

if (\_index == 0)

return \_allPhilosophers[\_allPhilosophers.Count - 1];

else

return \_allPhilosophers[\_index - 1];

}

}

public Philosopher RightPhilosopher

{

get

{

if (\_index == \_allPhilosophers.Count - 1)

return \_allPhilosophers[0];

else

return \_allPhilosophers[\_index + 1];

}

}

public void EatAll()

{

while (true)//(\_timesEaten < TIMES\_TO\_EAT)

{

this.Think();

this.TakePlace();

if (this.PickUp())

{

this.Eat();

this.PutDownLeft(); // Release chopsticks

this.PutDownRight();

this.FreePlace();

}

}

}

private void FreePlace()

{

\_serv.x--;

}

private void TakePlace()

{

\_serv.takePlace();

/\*lock (locker)

Console.WriteLine(this.Name + " " + \_serv.x);

{

while ( \_serv.x >= 4)

{

}

\_serv.x++;

}\*/

}

private bool PickUp()

{

if (Monitor.TryEnter(this.LeftChopstick)) // Try to pick up the left chopstick

{

Console.WriteLine(this.Name + " picks up left chopstick.");

if (Monitor.TryEnter(this.RightChopstick)) // Try to pick up the right

{

Console.WriteLine(this.Name + " picks up right chopstick.");

return true;

}

else

{

this.PutDownLeft();

}

}

return false; // Try again

}

private void Eat()

{

this.State = State.Eating;

\_timesEaten++;

Console.WriteLine(this.Name + " eats.");

Thread.Sleep(rnd.Next(10, 1000));

}

private void PutDownLeft()

{

Monitor.Exit(this.LeftChopstick);

Console.WriteLine(this.Name + " puts down left chopstick.");

}

private void PutDownRight()

{

Monitor.Exit(this.RightChopstick);

Console.WriteLine(this.Name + " puts down right chopstick.");

}

private void Think()

{

this.State = State.Thinking;

Thread.Sleep(rnd.Next(10, 1000));

}

}

public enum State

{

Thinking = 0,

Eating = 1

}

[DebuggerDisplay("Name = {Name}")]

public class Chopstick

{

private static int \_count = 1;

public string Name { get; private set; }

public Chopstick()

{

this.Name = "Chopstick " + \_count++;

}

}

}

# Приложение. Интерпретация временных диаграмм

