Отчет по лабораторной работе № 3

«Синхронизация процессов»

Выполнил: студент группы P3317

Плюхин Д.А.

Преподаватель: Лаздин Артур Вячеславович

# **Задание**

Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для* *моделирования передачи сообщений ввести специальные события, которые обозначают «А» , «B», «C» , «D» и конец сеанса для процессов* **Parent****и****Child.**

Процесс **Boss**:

запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;

запускает заданное количество процессов **Parent, Child**;

запрашивает кол-во сообщений, принятых от **Parent** **или** **Child**

принимает от каждого процесса **Parent, Child** сообщение и выводит сообщение и кто его отправил на консоль в одной строке. Принимать сообщение может **только от двух процессов** **Child и одного процесса Parent**,передача остальных сообщений от других процессов должнаблокироваться с помощью мьютексов;

завершает свою работу.

Процесс **Parent**:

* запрашивает с консоли сообщения, состоящее из *«А» , «B»* и передает их (по одному) процессу Boss;
* завершает свою работу.

Процесс **Child**:

• запрашивает с консоли сообщения, состоящее из *«C» , «D»* и передает их (по одному) процессу Boss;

* + завершает свою работу.

# **Исходный код**

**Файл child.cs**

using System;

using System.Threading;

namespace SecondLab

{

class Child

{

const string C\_EVENT\_NAME = "c\_event\_name";

const string D\_EVENT\_NAME = "d\_event\_name";

const string C\_R\_EVENT\_NAME = "c\_r\_event\_name";

const string D\_R\_EVENT\_NAME = "d\_r\_event\_name";

const string C\_MUTEX\_NAME = "c\_mutex\_name";

const string D\_MUTEX\_NAME = "d\_mutex\_name";

static void Main(string[] args)

{

EventWaitHandle c\_ewh = EventWaitHandle.OpenExisting(C\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle d\_ewh = EventWaitHandle.OpenExisting(D\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle c\_r\_ewh = EventWaitHandle.OpenExisting(C\_R\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle d\_r\_ewh = EventWaitHandle.OpenExisting(D\_R\_EVENT\_NAME+"\_"+args[0]);

Mutex mtc = Mutex.OpenExisting(C\_MUTEX\_NAME+"\_"+args[0]);

Mutex mtd = Mutex.OpenExisting(D\_MUTEX\_NAME+"\_"+args[0]);

char[] possible\_messages = {'C','D'};

string message\_codes\_sequence = "";

do{

Console.WriteLine("Please, type message codes sequence (only "+possible\_messages[0]+" and "+possible\_messages[1]+" are available):");

message\_codes\_sequence = getMessageCodesSequence(possible\_messages);

}while(message\_codes\_sequence == "");

foreach (char c in message\_codes\_sequence) {

if (c == 'C'){

c\_r\_ewh.WaitOne();

mtc.WaitOne();

c\_r\_ewh.Reset();

c\_ewh.Set();

mtc.ReleaseMutex();

} else {

d\_r\_ewh.WaitOne();

mtd.WaitOne();

d\_r\_ewh.Reset();

d\_ewh.Set();

mtd.ReleaseMutex();

}

}

}

static string getMessageCodesSequence(char[] possible\_messages){

string message\_codes\_sequence = Console.ReadLine();

foreach (char c in message\_codes\_sequence){

if (!Array.Exists(possible\_messages, element => element == c)) return "";

}

return message\_codes\_sequence;

}

}

}

**Файл parent.cs**

using System;

using System.Threading;

namespace SecondLab

{

class Parent

{

const string A\_EVENT\_NAME = "a\_event\_name";

const string B\_EVENT\_NAME = "b\_event\_name";

const string A\_R\_EVENT\_NAME = "a\_r\_event\_name";

const string B\_R\_EVENT\_NAME = "b\_r\_event\_name";

const string A\_MUTEX\_NAME = "a\_mutex\_name";

const string B\_MUTEX\_NAME = "b\_mutex\_name";

static void Main(string[] args)

{

EventWaitHandle a\_ewh = EventWaitHandle.OpenExisting(A\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle b\_ewh = EventWaitHandle.OpenExisting(B\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle a\_r\_ewh = EventWaitHandle.OpenExisting(A\_R\_EVENT\_NAME+"\_"+args[0]);

EventWaitHandle b\_r\_ewh = EventWaitHandle.OpenExisting(B\_R\_EVENT\_NAME+"\_"+args[0]);

Mutex mta = Mutex.OpenExisting(A\_MUTEX\_NAME+"\_"+args[0]);

Mutex mtb = Mutex.OpenExisting(B\_MUTEX\_NAME+"\_"+args[0]);

char[] possible\_messages = {'A','B'};

string message\_codes\_sequence = "";

do{

Console.WriteLine("Please, type message codes sequence (only "+possible\_messages[0]+" and "+possible\_messages[1]+" are available):");

message\_codes\_sequence = getMessageCodesSequence(possible\_messages);

}while(message\_codes\_sequence == "");

foreach (char c in message\_codes\_sequence) {

if (c == 'B'){

//Console.WriteLine("Sending B...");

b\_r\_ewh.WaitOne();

mtb.WaitOne();

b\_r\_ewh.Reset();

b\_ewh.Set();

mtb.ReleaseMutex();

} else {

a\_r\_ewh.WaitOne();

mta.WaitOne();

a\_r\_ewh.Reset();

a\_ewh.Set();

mta.ReleaseMutex();

}

}

}

static string getMessageCodesSequence(char[] possible\_messages){

string message\_codes\_sequence = Console.ReadLine();

foreach (char c in message\_codes\_sequence){

if (!Array.Exists(possible\_messages, element => element == c)) return "";

}

return message\_codes\_sequence;

}

}

}

**Файл main.cs**

using System;

using System.Threading;

using System.Diagnostics;

using System.Collections;

namespace SecondLab

{

class Boss

{

const string CHILD\_MUTEX\_NAME = "child\_mutex\_name";

const string PARENT\_MUTEX\_NAME = "parent\_mutex\_name";

const string A\_EVENT\_NAME = "a\_event\_name";

const string B\_EVENT\_NAME = "b\_event\_name";

const string C\_EVENT\_NAME = "c\_event\_name";

const string D\_EVENT\_NAME = "d\_event\_name";

const string A\_R\_EVENT\_NAME = "a\_r\_event\_name";

const string B\_R\_EVENT\_NAME = "b\_r\_event\_name";

const string C\_R\_EVENT\_NAME = "c\_r\_event\_name";

const string D\_R\_EVENT\_NAME = "d\_r\_event\_name";

const string A\_MUTEX\_NAME = "a\_mutex\_name";

const string B\_MUTEX\_NAME = "b\_mutex\_name";

const string C\_MUTEX\_NAME = "c\_mutex\_name";

const string D\_MUTEX\_NAME = "d\_mutex\_name";

private static Mutex msg\_quantity\_mutex;

private static int msg\_quantity;

private static Semaphore child\_handling\_semaphore;

private static Mutex parent\_handling\_mutex;

private static ArrayList aThreads;

private static ArrayList aThreadsInfo;

private static ArrayList bThreadsInfo;

private static ArrayList cThreadsInfo;

private static ArrayList dThreadsInfo;

private static ArrayList bThreads;

private static ArrayList cThreads;

private static ArrayList dThreads;

private static ArrayList runningThreads;

private static int parentIndex = 0;

private static int childIndex = 0;

static void crp(int tmp){

Thread parentHandlerThread = new Thread(() => parentHandler(tmp));

parentHandlerThread.Start();

}

static void Main()

{

aThreads = new ArrayList();

aThreadsInfo = new ArrayList();

bThreads = new ArrayList();

bThreadsInfo = new ArrayList();

cThreads = new ArrayList();

cThreadsInfo = new ArrayList();

dThreads = new ArrayList();

dThreadsInfo = new ArrayList();

runningThreads = new ArrayList();

child\_handling\_semaphore = new Semaphore(2, 2);

msg\_quantity\_mutex = new Mutex(false);

parent\_handling\_mutex = new Mutex(false);

int child\_quantity = getIntFromUser("How many child processes do you want?");

int parent\_quantity = getIntFromUser("How many parent processes do you want?");

msg\_quantity = getIntFromUser("How many messages do you want to get?");

int tmp;

for (int i = 0; i < child\_quantity; i++){

tmp = i;

Thread childHandlerThread = new Thread(() => childHandler(tmp));

childHandlerThread.Start();

}

for (int i = 0; i < parent\_quantity; i++){

tmp = i;

crp(tmp);

}

while (true){

SwitchHandledProcesses();

Thread.Sleep(30000);

}

}

static void ShiftChildIndex(){

childIndex += 1;

if (childIndex >= cThreads.Count){

childIndex = 0;

}

}

static void ShiftParentIndex(){

parentIndex += 1;

if (parentIndex >= aThreads.Count){

parentIndex = 0;

}

}

static void SwitchHandledProcesses(){

Console.WriteLine("Switching...");

foreach(Thread thread in runningThreads){

thread.Interrupt();

}

runningThreads = new ArrayList();

ArrayList runningThreadsInfo = new ArrayList();

runningThreadsInfo.Add(aThreadsInfo[parentIndex]);

runningThreadsInfo.Add(bThreadsInfo[parentIndex]);

ShiftParentIndex();

runningThreadsInfo.Add(cThreadsInfo[childIndex]);

runningThreadsInfo.Add(dThreadsInfo[childIndex]);

ShiftChildIndex();

runningThreadsInfo.Add(cThreadsInfo[childIndex]);

runningThreadsInfo.Add(dThreadsInfo[childIndex]);

ShiftChildIndex();

foreach(ArrayList info in runningThreadsInfo){

EventWaitHandle arg0 = (EventWaitHandle)info[0];

EventWaitHandle arg1 = (EventWaitHandle)info[1];

Mutex arg2 = (Mutex)info[2];

Thread thread = new Thread(() => fEventHandler(ref arg0, ref arg1, ref arg2, (int)info[3], (string)info[4], (string)info[5]));

runningThreads.Add(thread);

thread.Start();

}

}

static void parentHandler(int id){

Console.WriteLine("Parent "+id+" is created");

EventWaitHandle ewh\_a = new EventWaitHandle(false, EventResetMode.ManualReset, A\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_b = new EventWaitHandle(false, EventResetMode.ManualReset, B\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_r\_a = new EventWaitHandle(true, EventResetMode.ManualReset, A\_R\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_r\_b = new EventWaitHandle(true, EventResetMode.ManualReset, B\_R\_EVENT\_NAME+"\_"+id);

Mutex mta = new Mutex(false, A\_MUTEX\_NAME+"\_"+id);

Mutex mtb = new Mutex(false, B\_MUTEX\_NAME+"\_"+id);

aThreadsInfo.Add(new ArrayList{ewh\_a, ewh\_r\_a, mta, id, "A", "parent"});

Thread aEventHandlerThread = new Thread(() => fEventHandler(ref ewh\_a, ref ewh\_r\_a, ref mta, id, "A", "parent"));

aThreads.Add(aEventHandlerThread);

bThreadsInfo.Add(new ArrayList{ewh\_b, ewh\_r\_b, mtb, id, "B", "parent"});

Thread bEventHandlerThread = new Thread(() => fEventHandler(ref ewh\_b, ref ewh\_r\_b, ref mtb, id, "B", "parent"));

bThreads.Add(bEventHandlerThread);

Process process\_parent = new Process();

process\_parent.StartInfo.FileName = "parent.exe";

process\_parent.StartInfo.Arguments = ""+id;

process\_parent.Start();

process\_parent.WaitForExit();

}

static void fEventHandler(ref EventWaitHandle ewh, ref EventWaitHandle rewh, ref Mutex mt, int id, string msg\_type, string proc\_type){

int lok = 10;

try{

while(true){

ewh.WaitOne();

Console.WriteLine("Got message "+msg\_type+" from "+id+" "+proc\_type+"!");

mt.WaitOne();

ewh.Reset();

rewh.Set();

mt.ReleaseMutex();

msg\_quantity\_mutex.WaitOne();

msg\_quantity -= 1;

lok = msg\_quantity;

msg\_quantity\_mutex.ReleaseMutex();

if (lok < 1) {

Console.WriteLine("Maximal number of messages reached. Exiting...");

Environment.Exit(0);

}

}

} catch(ThreadInterruptedException e)

{

}

}

static void childHandler(int id){

EventWaitHandle ewh\_c = new EventWaitHandle(false, EventResetMode.AutoReset, C\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_d = new EventWaitHandle(false, EventResetMode.AutoReset, D\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_r\_c = new EventWaitHandle(true, EventResetMode.ManualReset, C\_R\_EVENT\_NAME+"\_"+id);

EventWaitHandle ewh\_r\_d = new EventWaitHandle(true, EventResetMode.ManualReset, D\_R\_EVENT\_NAME+"\_"+id);

Mutex mtc = new Mutex(false, C\_MUTEX\_NAME+"\_"+id);

Mutex mtd = new Mutex(false, D\_MUTEX\_NAME+"\_"+id);

cThreadsInfo.Add(new ArrayList{ewh\_c, ewh\_r\_c, mtc, id, "C", "child"});

Thread cEventHandlerThread = new Thread(() => fEventHandler(ref ewh\_c, ref ewh\_r\_c, ref mtc, id, "C", "child"));

cThreads.Add(cEventHandlerThread);

//cEventHandlerThread.Start();

dThreadsInfo.Add(new ArrayList{ewh\_d, ewh\_r\_d, mtd, id, "D", "child"});

Thread dEventHandlerThread = new Thread(() => fEventHandler(ref ewh\_d, ref ewh\_r\_d, ref mtd, id, "D", "child"));

dThreads.Add(dEventHandlerThread);

//dEventHandlerThread.Start();

Process process\_child = new Process();

process\_child.StartInfo.FileName = "child.exe";

process\_child.StartInfo.Arguments = ""+id;

process\_child.Start();

//process\_child.WaitForExit();

}

static uint getUintFromUser(string msg){

Console.WriteLine(msg);

return Convert.ToUInt32(Console.ReadLine());

}

static int getIntFromUser(string msg){

Console.WriteLine(msg);

return Convert.ToInt32(Console.ReadLine());

}

}

}

# **Результат**

How many child processes do you want?

4

How many parent processes do you want?

5

How many messages do you want to get?

20

Switching...

Got message C from 1 child!

Got message D from 1 child!

Got message C from 1 child!

Got message B from 0 parent!

Got message A from 0 parent!

Got message A from 0 parent!

Got message B from 0 parent!

Switching...

Got message A from 1 parent!

Got message B from 1 parent!

Got message B from 1 parent!

Got message A from 1 parent!

Got message C from 2 child!

Got message D from 2 child!

Got message D from 3 child!

Got message C from 3 child!

Got message C from 3 child!

Got message D from 3 child!

Switching...

Got message B from 2 parent!

Got message B from 2 parent!

Got message B from 2 parent!

Maximal number of messages reached. Exiting...

# **Вывод**

Таким образом, была реализована относительно простая модель межпроцессорного взаимодействия.