Университет ИТМО Кафедра ИПМ

Отчет по лабораторной работе № 3

«Синхронизация процессов»

Выполнил: студент группы P3317

Плюхин Д.А.

Преподаватель: Лаздин Артур Вячеславович

Санкт-Петербург
2017 год

Задание

Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события, которые обозначают «A» , «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 должна блокироваться с помощью мьютексов;

завершает свою работу.

Процесс **Parent**:

- запрашивает с консоли сообщения, состоящее из *«A» , «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.