10 Дәріс. Тізбектерді синхронизациялау

10.1 Әрекет жөніндегі түсінік

Тізбектің контекстін өзгертетін командалардың кез келген реттілігі әрекет деп аталады. Тізбек контекстінің анықтамасы 2 дәрісте қарастырылады – бұл тізбек өзінің орындалуы кезінде жүгіне алатын компьютер жадысының облысы.

Әрекет өзінің орындалуы кезінде үзілмесе, және әрекеттің өзімен ғана өзгертілсе,онда әрекет үзіліссіз (үздіксіз) деп аталады.

Әрекет тек микропроцессордың үзу сигналымен ғана үзілуі мүмкін.

Бірінші талап тек бірпроцессорлық жүйелерге ғана әділ болады, оларда әрекеттің орындау уақытына үзуге де тыйым салуға болады.

Мультипроцессорлық жүйелер үшін әрекеттер әртүрлі тізбектермен параллельді орындалуы да және де бір-бірімен қиылысуы да мүмкін. Сондықтанда мультипроцессорлық жүйелер үшін әрекеттердің үзіліссіздігі бір процессорда орындалатын әрекетке басқа процессорда орындалып жатқан әрекетті өзгертуге тыйым салумен қамтамасыз етіледі.

Сонымен, мультипроцессорлық жүйелерде әрекеттердің үзіліссіздігін қамтамасыз етудің маңызды міндеті параллельді тізбектер жұмысының келістіру (үйлестіру).

Осы міндетті шешудің бір нұсқаларының бірі тізбектердің жұмысын синхронддау.

Жалпы алғанда, тізбектер өзара алмасатын сигналдардың көмегімен оларды орындаудың қандай да бір бекітілген реттілігіне қол жеткізу тізбектерді синхрондау деп аталады.

Синхрондау болжанылған нәтижені алу үшін тізбектердің әрекеттерін координациялау (реттеу, үйлестіру) [Албахари б.739.]

10.2 Тізбектерді синхрондау құралдарының жіктелуі

Осы дәрісте бір үдерісте тізбектерді синхрондау мысаларындағы синхрондаудың теориялық аспектілерін қарастырамыз.

Әдетте, бағдарламаның негізгі әдісі (Main) басты тізбек деп аталады, ал басты тізбек үшін кейбір міндеттерді шешетін әдістер қосымша немесе екінші тізбектер деп аталады.

Әрине, осы тізбектердің жұмысы реттелуге тиіс. Тізбектер жұмысының реттелгендігінен басқа реалды (айқын) (оқуға емес) бағдарламалар кейбір деректерді бірлесе пайдалану міндеттерін шешуге тиіс (тізбектерге деректерді және тізбектер арасында деректерді жіберудің жеке сұрақтарын біз алдыңғы дәрістерде қарастырдық), бірақ синхрондау теорияларын зерделеу үшін осы міндеттерді жеке-жеке қарастырған тиімді.

«Барлық синхрондау құралдарын шартты түрде төр санатқа бөлуге болады:

- тізбектерді синхрондаудың қарапайым құралдары, мұнда глобальді (ауқымды) айнымалылар және тізбектің орындалуын тоқтатудың қарапайым әдістері (Sleep және Join) жатады;

- арнайы блоктаушы конструкциялар,мұнда синхрондау объектілері (Mutex және Semaphore) және lock операторы жатады;

- арнайы сигналдарды беруге және қабылдап алуға негізделген конструкциялар. Осы конструкциялар басқа тізбектен сигнал алғанға дейін тізбектің орындалуын тоқтатып тұрады. Әдетте, сигналдық механизмді пайдаланатын екі осындай конструкциялар қолданылады, олар оқиғаларды күту дескрипторлары және Monitor класының Wait/Pulse әдістері;

- тізбектердің орындалуын тоқтатуды талап етпейтін құралдар (бөгет етпейтін (кідіртпейтін) синхрондау құралдары), оларға Interlocked класы және volatile арнайы кілт сөзі, ол деректерді оқу –жазу операцияларын кэштеуге тыйым салуға мүмкіндік береді.»[Албахари б. 739-740].

Тізбектерді синхрондаудың аталған құралдарынан басқа, ContextBoundObject класымен және Synchronization атрибутымен жүзеге асырылатын тізбектің ресурстарына қатынауды автоматты синхрондау (блоктау) нұсқасы бар. Оларды біз бесінші санат құралдары деп атаймыз.

Синхрондаудың қандай да бір құралдарын таңдау шешілетін міндетпен анықталады және бағдарламашыға байланысты.

Синхронда құрадларының зертелуін ең қарапайым санаттардан бастаймыз.

Албахари Интернетте жарияланған мақаларының бірінде тізбектерді синхрондау құралдарының келесі тізімін келтіреді (мақала аудармашысы оларды ағындар деп атайды):

«Тізбектерді синхрондаудың маңызды құралдары»

Келесі кестелерде ағындарды кординациялау (синхрондау) үшін .NET инструменттары келтірілген:

Простейшие методы блокировки

Конструкция	**Назначение**
Sleep	Блокировка на указанное время
Join	Ожидание окончания другого потока

Блокировочные конструкции

Конструкция	**Назначение**	**Доступна из других процессов?**	**Скорость**
Lock	Гарантирует, что только один поток может получить доступ к ресурсу или секции кода. Уақыттың әр мезетінде оны тек бір ғана тізбек қолдана алатындай етіп, код секциясын блоктайды	нет	быстро
Mutex	Гарантирует, что только один поток может получить доступ к ресурсу или секции кода. Может использоваться для предотвращения запуска нескольких экземпляров приложения.	да	средне
Semaphore	Гарантирует, что не более заданного числа потоков может получить доступ к ресурсу или секции кода.	да	средне

(Для автоматической блокировки также могут использоваться контексты синхронизации.)

Конструкция	Назначение	Доступна из других процессов?	Скорость
EventWaitHandle	Позволяет потоку ожидать сигнала от другого потока.	да	средне
Wait and Pulse*	Позволяет потоку ожидать, пока не выполнится заданное условие блокировки.	нет	средне

Сигнальные конструкции

Конструкция	Назначение	Доступна из других процессов?	Скорость
Interlocked*	Выполнение простых не блокирующих атомарных операций.	Да – через разделяемую память	очень быстро
volatile*	Для безопасного не блокирующего доступа к полям.	Да – через разделяемую память	очень быстро

10.3 Тізбектерді синхрондаудың қарапайым құралдары

Тізбектерді синхрондаудың теорияларын қарастыру кезінде, дәстүрлі түрде олардың жұмысының кезектілігін көзбен бақылау үшін кейбір әртүрлі мәндердің циклдық шығарылуы қолданылады, мысалы, әртүрлі символдар немесе сандарды шығару. Дәстүрден алшақтамайық, біздің тізбектердің жұмысы да монитор экранның консолдық терезесіне сандық мәндерді шығаруда болады (реалды тізбектер басты тізбек үшін жұмыс атқарады).

Басты тізбектен басқа, біздің бағдарламада қосымша екі тізбек бар деп болжайық. Басты тізбек консолдық терезеге 0-ден 9-ға дейін, бірінші қосымша тізбек –10-нан 19-ға дейін, ал екінші қосымша тізбек –20-дан 2-ға дейін сандарды шығарсын (олардың жұмысы осындай).

Тізбектері синхрондалмаған, глобальді айнымалының көмегімен және бір үдеріс тізбектерін синхрондау құралдары - **Sleep, Join**и **lock** әдістерінің көмегімен синхрондалған бірнеше оқу бағдарламаларын қарастырайық.

Синхрондалмаған тізбектері бар бағдарламада бірінші тізбек өз класының ішінде жарияланған, ал екінші тізбек – статикалық әдіспен көрсетілген.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Not_Cinxr
    {
        class Pervaj_1
        {
            public void Niti_1()
            {
                for (int i = 10; i < 20; i++)
                    Console.Write(i + " ");
            }
```

```
        }
        static void Niti_2()
        {
            for (int i = 20; i < 30; i++)
                Console.Write(i + " ");
        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            for (int i = 0; i < 10; i++)
                Console.Write(i + " ");
            Console.ReadLine();
        }
    }
}
```

Работа программы:
0 20 21 10 11 12 13 14 15 16 17 18 19 1 2 3 4 5 6 7 8 9 22 23 24 25 26 27 28 29
Работа нитей не синхронизирована – вывод осуществляется «вперемешку».

10.3.1 Ауқымды (Глобальді) айнымалыны пайдалану

Flag глобальді айнымалысының көмегімен тізбектерді синхрондау мысалын қарастырайық. Барлық тізбектер жұмысқа қосылады, бірақ тізбектердің «денелері» ауқымды айнымалыдан тізбек жұмысына рұқсат «тосатын» **goto** және **if** операторларымен блокталынады.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Cin_Gl
    {
        public static int flag = 0;
        class Pervaj_1
        {
            public void Niti_1()
            {
               m1: if (flag == 0)
                 {
                     for (int i = 10; i < 20; i++)
                         Console.Write(i + " ");
                     flag++;
                 } else goto m1;
            }
        }
        static void Niti_2()
        {
          m1:  if (flag == 2)
            {
                for (int i = 20; i < 30; i++)
```

```
                    Console.Write(i + " ");
                flag++;
            } else goto m1;
        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
        m1: if (flag == 1)
            {
                for (int i = 0; i < 10; i++)
                    Console.Write(i + " ");
                flag++;
            }
            else goto m1;
            Console.ReadLine();
        }
    }
}
```

Работа программы:

10 11 12 13 14 15 16 17 18 19 0 1 2 3 4 5 6 7 8 9 20 21 22 23 24 25 26 27 28 29

Сначала работает первая нить(**Flag** = 0), затем главная нить (**Flag** = 1) и затем работает вторая нить (**Flag** = 2).

10.3.2 Sleep әдісін пайдалану

Ағымдағы тізбекті берілген миллисекундқа блоктайтын Thread.Sleep() әдісінің көмегімен тізбектерді синхрондау мысалын қарастырайық.

Thread.Sleep(0) әдісі – бөлінген уақыт квантынан бас тартады.

Thread.Sleep(100) әдісі –тізбектің орындалуын 100 миллисекундқа блоктайды –» 100 миллисекундқа «ұйықтау».

```
Thread.Sleep(0);   // отказаться от одного кванта времени CPU
  Thread.Sleep(1000);                  // заснуть на 1000 миллисекунд
  Thread.Sleep(TimeSpan.FromHours(1)); // заснуть на 1 час
  Thread.Sleep(Timeout.Infinite);      // заснуть до прерывания
```

Тізбек блоктаушы конструкцияларын пайдаланған соң блоктау аяқталғанша дейін уақыт кванттарын алуын тоқтататынын атап кету керек.

Мысалда бағдарламаның консолдық терезесіне әр шығудан кейін тізбектердің блокталуы жүзеге асырылған.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Cin_sleep
    {
        public static int flag = 0;
        class Pervaj_1
        {
```

```
            public void Niti_1()
            {
                for (int i = 10; i < 20; i++)
                { Console.Write(i + " "); Thread.Sleep(1); }
            }
        }
        static void Niti_2()
        {
                for (int i = 20; i < 30; i++)
                { Console.Write(i + " "); Thread.Sleep(1); }

        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            for (int i = 0; i < 10; i++)
            { Console.Write(i + " "); Thread.Sleep(1); }

            Console.ReadLine();
        }
    }
}
```

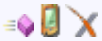
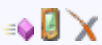
Работа программы:
0 20 10 21 1 11 2 12 22 13 23 3 4 24 14 25 5 15 6 26 16 27 17 7 18 28 8 19 9 29

Бағдарламаны тексеру барысында Sleep() әдісінің кідірту уақыты 1-ден 150 миллисекундқа дейін өзгерді. Бағдарлама жұмысының барлық нәтижелерінде шығару «үштікпен» жүргізіледі, бірақ «үштік» ішінде кезек сақталмады. Сонымен, тізбек кезекті шығарудан соң,басқа тізбектің жұмысына (тек бір шығаруға) рұқсат беріп қандай да бір уақытқа «ұйықтайды».

10.3.3 Join() әдісін пайдалану

Ағымдағы тізбектің аяқталуына дейін шақырушы тізбекті блоктайтын Join() әдісінің көмегімен тізбектерді синхрондаудың мысалын қарастырамыз. Сонымен, басқа тізбек аяқталғанға дейін Join() әдісімен блоктап тастауға болады.

Join() әдісін пайдаланудың үш нұсқасы бар:

	Имя	Описание
	Join()	Блокирует вызывающий поток до завершения потока, продолжая отправлять стандартные сообщения COM и SendMessage.
	Join(Int32)	Блокирует вызывающий поток до завершения потока или истечения указанного времени, продолжая отправлять стандартные сообщения COM и SendMessage.

	Join(TimeSpan)	Блокирует вызывающий поток до завершения потока или истечения указанного времени, продолжая отправлять стандартные сообщения COM и SendMessage.

Мысалда Join() әдісімен блоктаудың түрлі нұсқалары жүзеге асырылған («//» сәйкес әдістерді қосу немесе өшіру).

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Cin_sleep
    {
        public static int flag = 0;
        class Pervaj_1
        {
            public void Niti_1()
            {
                for (int i = 10; i < 20; i++)
                { Console.Write(i + " "); Thread.Sleep(10); }
            }
        }
        static void Niti_2()
        {
                for (int i = 20; i < 30; i++)
                { Console.Write(i + " "); Thread.Sleep(10); }

        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            t1.Join();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            t2.Join();
            for (int i = 0; i < 10; i++)
            { Console.Write(i + " "); Thread.Sleep(10); }

            Console.ReadLine();
        }
    }
}
```

Работа программы – только **`t1.Join();`**:
10 11 12 13 14 15 16 17 18 19 0 20 1 21 22 2 23 3 4 24 25 5 6 26 27 7 28 8 9 29
Работа программы – только **`t2.Join();`**:
10 20 11 21 12 22 13 23 14 24 15 25 16 26 17 27 18 28 29 19 0 1 2 3 4 5 6 7 8 9
Работа программы – и **`t1.Join();`** и **`t2.Join();`**:

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 0 1 2 3 4 5 6 7 8 9

Бірінші жағдайда бірінші тізбек екінші және басты тізбектердің орындалуын «блоктайды», олар блоктау аяқталған соң синхронды емес болып жұмыс істейді.

Екінші жағдайда тек басты тізбек «блокталынады», ал бірінші және екінші тізбектер синхронды емес болып жұмыс істейді.

Үшінші жағдайда басында бірінші тізбек екінші және бастапқы тізбектердің орындалуын «блоктайды», бұдан соң екінші тізбек басты тізбекті «блоктайды».

Блоктаудың осы құралы тізбектердің жұмысын ретке келтіруге мүмкіндік береді.

10.3.4 **lock** операторын пайдалану

Lock операторы **Mutex** және **Semaphore** синхрондау объектілерімен бірге арнайы блоктаушы конструкцияларға жатады**.**

Әртүрлі үдерістердің тізбектерін блоктайтын **Mutex** және **Semaphore** синхрондау объектілеріне қарағанда, **lock** операторы тек бір үдерістің ішінде ғана тізбектердің жұмысын блоктауға мүмкіндік береді. Сондықтанда **lock** операторының жұмысын осы дәрісте қарастырамыз.

Алдыңғы дәрісте біз тізбектерге деректерді жіберу үшін **lock** операторының жұмысын қарастырған болатынбыз. Осы мысалда y = a*x + b/x + c өрнегін есептеу үшін үш тізбекті қосамыз. Бірінші тізбек c мәнін алады және тиісінше екінші және үшінші тізбектерде орындалатын a*x және b/x есептеулердің нәтижелерін тосады. x айнымалысы бағдарламада глобальді айнымалы ретінде жарияланады.

Оқу мақсатында бүкіл бағдарламаның жүзеге асырылуын 4 файл ретінде орындаймыз – бағдарлама файлы және тізбектер іске қосылатын әдістер үшін болатын кластар файлынан үш файл болады. Бағдарламаның бірінші файлының бастапқы коды келесідей түрде болады:

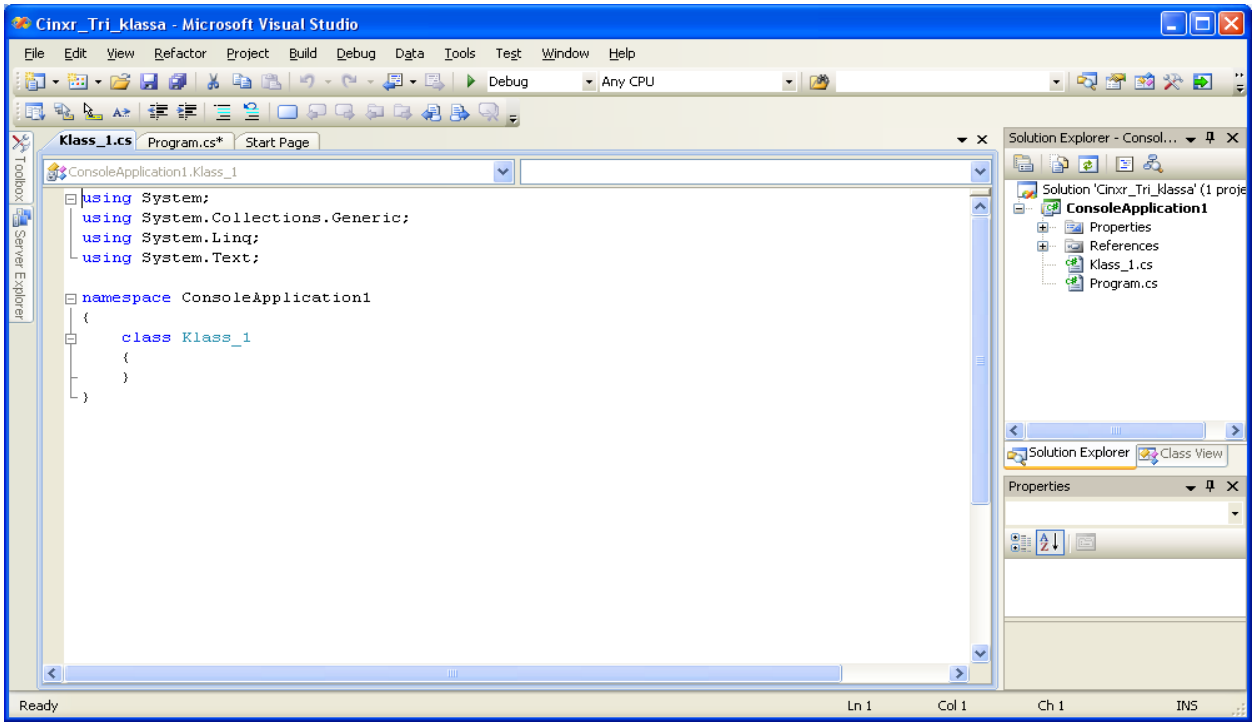
```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Program
    {
        public static double rez,x;

        static void Main()
        {
            Console.WriteLine("Введите x");
            x = double.Parse(Console.ReadLine());
            Klass_1 Thr1 = new Klass_1();
            Thr1.thread1.Join();

            Console.ReadKey();
        }
    }
}
```

Жобаға Klass_1 файлын құру және оны қосу үшін Project режімінде AddClass командасын таңдап алу және пайда болған терезеде Klass_1 деген құрылатын файлдың атауын көрсету қажет. Орта Klass_1.cs файлын құрады, оны жобаға қосады және файл бағдарламасының кодын редакциялау режіміне ауыстырады (10.1 суретін қараңыз).



10.1-сурет –Klass_1.cs файлын құру және жобаға қосу

Klass_1.cs файлында диалог режіміне **c** айнымалысының мәнін енгіземіз және жұмысқа жобаның екінші тізбегін қосамыз, оның жұмыс нәтижесін бүкіл өрнекті есептеу үшін пайдаланатын боламыз.

Klass_1.cs файлының бастапқы коды:

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
 class Klass_1
 {
  private double c;
  public Thread thread1;
  public Klass_1()
  {
   thread1 = new Thread(Rab);
   thread1.Start();
  }
  public void Rab()
  {
   lock (this)
   {
    Console.WriteLine("Нить 1 стартовала");
    Console.WriteLine("Введите C");
    c = double.Parse(Console.ReadLine());
    Klass_2 Thr2 = new Klass_2();
```

```
    Thr2.thread2.Join();
    Program.rez = Program.rez + c;
    Console.WriteLine("Нить 1 завершила работу. Результат =" +
Program.rez);
   }
  }
 }
}
```

Кодтан бірінші тізбектің жұмысын аяқтау үшін екінші тізбек жұмысының (екінші кластың әдісімен жұмыс істейтін) нәтижесі қажетті екенін көреміз. Klass_2.cs файлын құрамыз және оны жобаға қосамыз.

Klass_2.cs файлында диалог режіміне b айнымалысының мәнін енгіземіз, b/x есептейміз және жұмысқа жобаның үшінші тізбегін қосамыз, оның жұмыс нәтижесін бүкіл өрнекті есептеу үшін пайдаланамыз.

Klass_2.cs файлы келесі бастапқы кодқа ие:

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
 class Klass_2
 {
  private double b;
  public Thread thread2;
  public Klass_2()
  {
   thread2 = new Thread(Rab);
   thread2.Start();
  }
  public void Rab()
  {
   lock (this)
   {
    Console.WriteLine("Нить 2 стартовала");
    Console.WriteLine("Введите B");
    b = double.Parse(Console.ReadLine());
    Klass_3 Thr3 = new Klass_3();
    Thr3.thread3.Join();
    Program.rez = Program.rez + b/Program.x;
    Console.WriteLine("Нить 2 завершила работу. Результат =" +
Program.rez);
   }
  }
 }
}
```

Klass_3.cs файлында диалог режіміне a айнымалысының мәнін енгіземіз, a*x есептейміз және жобаның үшінші тізбегінің жұмысын аяқтаймыз. Оның жұмысының аралық нәтижесін монитор экранына шығарамыз.

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
```

```
 class Klass_3
 {
  private double a;
  public Thread thread3;
  public Klass_3()
  {
   thread3 = new Thread(Rab);
   thread3.Start();
  }
  public void Rab()
  {
   lock (this)
   {
    Console.WriteLine("Нить 3 стартовала");
    Console.WriteLine("Введите A");
    a = double.Parse(Console.ReadLine());
    Program.rez = Program.rez + a * Program.x;
    Console.WriteLine("Нить 3 завершила работу. Результат =" +
Program.rez);
   }
  }
 }
}
```

Работа программы:
Введите x
2,5
Нить 1 стартовала
Введите C
3
Нить 2 стартовала
Введите B
5
Нить 3 стартовала
Введите A
3
Нить 3 завершила работу! Значение равно: 7,5
Нить 2 завершила работу! Значение равно: 9,5
Нить 1 завершила работу! Значение равно: 12,5

Бағдарлама жұмысының нәтижелері бойынша x айнымалысының мәні енгізілген соң бірінші тізбек қосылатыны, бұдан соң екінші тізбек және кейіннен үшінші тізбек қосылатынын көреміз. Тізбектер жұмысының аяқталуы кері ретпен жүргізіледі. Тізбектер жұмысын синхрондау lock операторының көмегімен және кезекті тізбекті алдыңғы тізбектің тізбектік функциясынан қосумен қол жеткізіледі.