7 Жүйелік бағдарламалаудағы тізбек (Thread, нить) түсінігі

7.1 Тізбек туралы түсінік

Компьютер процессорының жұмысы бағдарлама компиляциясынан кейін қалыптасатын командаларды (нұсқаулықтарды) орындау болып табылады. Бұл бағдарлама нұсқаулықтарының тізбектілігі бағдарламаны басқару командаларының ағыны деп аталады. «Басқару ағыны» термині C++ тілінде пайдаланылады. C# тілінде бағдарламаны басқару командаларының ағыны тізбек деп алынған - компьютер процессоры атқаратын нұсқаулықтар «тізбектелген» тізбекпен (thread) салыстырады.

Егер операциялық жүйеде бір тізбек ғана бар болса, онда операциялық жүйе бір бағдарламалық деп аталады.

Егер операциялық жүйеде бір мезетте әртүрлі бағдарламалардың бірнеше тізбектері болса, онда ол мультибағдарламалық деп аталады.

Компьютер жұмысының тиімділігі көбінесе компьютер процессорының жүктемесімен анықталады. Сондықтанда, мультибағдарламалық ОЖ маңызды міндеті процессордың тиімді жүктемесін қамтамасыз ететіндей етіп, бағдарламалар тізбектерін диспетчерлеу.

Бағдарламаның кез келген тәуелсіз орындалатын фрагменті бағдарлама тізбегі бола алады.

a және b сандарының қосындысын есептейтін бағдарлама мысалын қарастырайық.

```
using System;
namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   int a,b,c;
   string buf;
   Console.Write("Введите целое значение a ");
   buf = Console.ReadLine();
   a = Convert.ToInt32(buf);
   Console.Write("Введите целое значение b ");
   buf = Console.ReadLine();
   b = Convert.ToInt32(buf);
   c = a + b;
   Console.WriteLine("a+b={0}", c);
   Console.WriteLine("Для продолжения нажмите клавишу Enter");
   Console.ReadLine();
  }
 }
}
```

Осы бағдарламаның әр әрекеті әрдайым бір мағыналы анықталған және бірінен кейін бірі орындалады – бағдарламада тек бір ғана тізбек бар.

a және b сандарының қосындысын есептейтін функцияны енгізу арқылы бағдарламаны өзгертеміз.

```
using System;
namespace ConsoleApplication1
{
 class Program
 {
  public static int sum(int a, int b)
  { return a + b; }
  static void Main(string[] args)
  {
   int a, b, c;
   string buf;
   Console.Write("Введите целое значение a ");
   buf = Console.ReadLine();
   a = Convert.ToInt32(buf);
   Console.Write("Введите целое значение b ");
   buf = Console.ReadLine();
   b = Convert.ToInt32(buf);
   c = sum(a,b);
   Console.WriteLine("a+b={0}", c);
   Console.WriteLine("Для продолжения нажмите клавишу Enter");
   Console.ReadLine();
  }
 }
}
```

Егерде бағдарламаны орындау кезінде **Main** функциясы **sum** функциясының орындалуын тосса, оонда бағдарлама бір тізбекпен қалады.

Егерде бағдарламаны орындау кезінде **Main** функциясы **sum** функциясының орындалуын тоспай, орындалауын жалғастыра берсе, онда бағдарламада екі тізбек пайда болады – басты, **Main** функциясының тізбегі және **sum** функциясының тізбегі. Осы жағдайда **Main** функциясымен монитор экранына шығарылған **c** айнымалысының мәні **a және b** айнымалылардың қосындысына сәйкес келмеуі мүмкін.

Шартты өтудің операторларын пайдаланатын бағдарламаларда бұдан да көп тізбек болуы мүмкін.

Бағдарламаның жұмысын дұрыс ұйымдастыру үшін тізбектер арасында деректерді беруді, деректерді бірігіктіріп пайдалануды және әртүрлі тізбектердің немесе тізбектік функциялардың жұмысын синхрондауды (ретке келтіру) үйрену керек.

7.2 Параллельді тізбектермен пайдаланылатын әдістерге қойылатын талаптар

Параллельді тізбектермен «қауіпсіз» шақыруға болатын әдістер қандай талаптарды қанағаттандыруы керектігін қарастырамыз.

Бұл үшін тізбек контексті түсінігін - тізбек өзінің орындалуы кезінде өзі жүгіне алатын компьютер жадысының облысы ретінде анықтаймыз.

Келесі функцияның жұмысын қарастырамыз:

```
public static int funk1(int n)
{
  if (n >= 0) n --;
  if (n  < 0) n ++;
  return n;
}
```

Осы функцияны шақыру кезінде **n** айнымалының көшірмесі құрылады. Осы көшірме шартқа сәйкес өзгертіледі және нәтижесі тізбекке қайтарылады. Әрі қарай айнымалының көшірмесі жадыдан жойылады. Осындай функция тізбек үшін қауіпсіз деп аталады.

funk1 функциясын ол **n** ауқымды айнымалысымен жұмыс істей алатындай етіп өзгертеміз.

```
int n;
. . .
public static void funk2()
{
  if (n >= 0) n --;
  if (n  < 0) n ++;
}
```

funk2 функциясын параллельді түрде бірнеше тізбек шақыртса **n** айнымалысының мәні орынсыз өзгеруі мүмкін (**n** айнымалысы бірден көп өзгертілуі орын алуы мүмкін). Осындай функциялар тізбек үшін қауіпсіз емес деп аталады.

Бағдарламалаудың кейбір тілдерінде, мысалы, C++ тілінде, функцияның статикалық айнымалылары түсінігі бар. Статикалық айнымалылар функциялар ішінде (жергілікті айнымалылар) жарияланады, бірақ осы айнымалылардың мәні функция жұмысының соңында жадыда сақталынады және функцияның қайта шақыртуда пайдаланылуы мүмкін. Яғни осындай статикалық айнымалы функция ішінде бағдарламаның ауқымды айнымалысы сияқты әрекет етеді. Бұл сияқты функцияға параллельді түрде бірнеше тізбек қатынаса, онда статикалық айнымалының мәні қандай да бір тізбектің функцияларын шақырту санына сәйкес келмеуі мүмкін.

Жалпы жағдайда функция келесі талаптарды қанағаттандырса, қауіпсіз немесе реентерабелді деп аталады:

- мәндері параллельді жұмыс істеп тұрған тізбектермен өзгертілетін ауқымды айнымалыларды пайдаланбаса;
- функция ішінде анықталған және қатарлас жұмыс істеп тұрған тізбектермен өзгертілетін статикалық айнымалыларды пайдаланбаса;
- функция ішінде анықталған статикалық деректерге нұсқағыштарды қайтармаса.

Егерде, тізбектер бірігіп қолданатын, әртүрлі тізбектер функцияларының ресурстарына блоктаудың әртүрлі тәсілдерін пайдаланса, қауіпсіз функцияларға қойылатын аталған талаптарды қамтамасыз етуге болады.

7.3 Тізбектер күйі

Тізбектер күйі компьютер процессорының тізбектерден тұратын бағдарламамен жұмыс істеуге дайындығына және бағдарламаның өз дайындығына тәуелді (байланысты) болады.

Процессордың дайындығы оның осы бағдарламаны орындауға «бөлуімен» (айырықшалауымен) анықталады – яғни процессор осы бағдарламамен жұмыс үшін уақыт квантын алады.

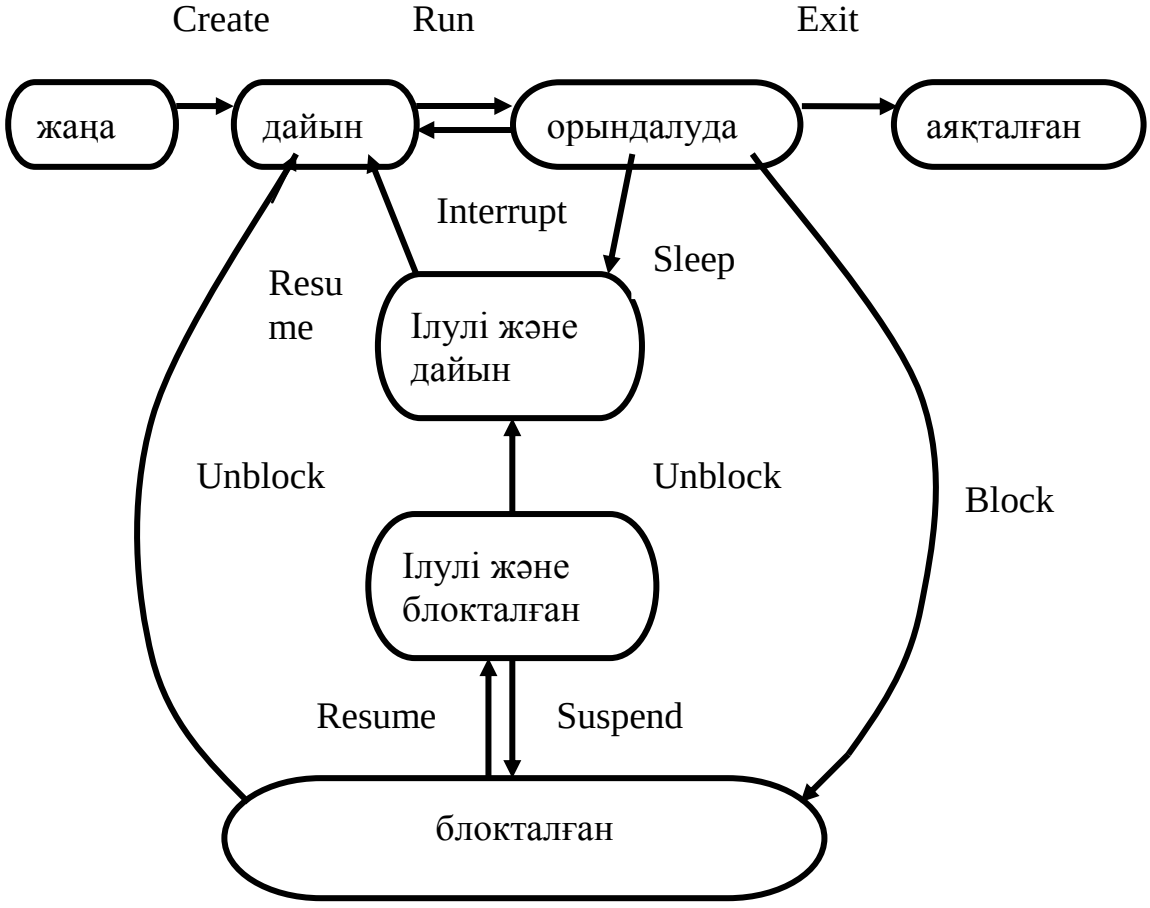
Бағдарламаның дайындығы оның бағдарламаны орындау үшін қажетті барлық ресурстарды алуымен байланысты. Процессор мен бағдарламаға байланысты ағынның келесі күйлері мүмкін болуы мүмкін:

- тізбек орындауға дайын («бөлінбеген» (жеке алынған), «дайын»);
- тізбек блокталған («бөлінбеген», «дайын емес»);
- тізбек орындалуда («бөлінген», «дайын»).

Әдетте, тізбектің аталған күйлеріне тағы екі күй қосылады – жаңа (тізбек құрылуда, бірақ дайын емес) және аяқталған (тізбек жұмысының аяқталуына сәйкес келетін күй).

Жұмысы өзара байланысты болатын бірнеше тізбектен тұратын бағдарламаларда тағы да екі тізбек болады –тізбектің орындалуын тоқтату және тізбектің орындалуын жаңғырту .

Тізбектің барлық аталған күйлерін ескере отырып, оның реттілігінің ***(поведения)*** келесі моделін салуға болады.



7.1-сурет – Тізбектің жеті күйінің моделі.

Create операциясы тізбекті «жаңа» күйінен «дайын» күйіне ауыстырады.

Run операциясы тізбекті «дайын» күйінен «орындалуда» күйіне ауыстырады - тізбекке процессорлық уақыт бөлінеді.

Exit операциясы тізбекті «орындалуда» деген күйінен «аяқталды» күйіне ауыстырады.

Interrupt операциясы тізбекті «орындалуда» деген күйінен «дайын» күйіне ауыстырады - әдетте осы операция тізбектің орындалуына бөлінген процессорлық уақыт аяқталғанда тізбек үстінен атқарылады.

Resume операциясы тізбектің орындалуын қайта қалыпқа келтіреді.

Sleep операциясы тізбектің орындалуын тоқтатып қояды.

Unblock операциясы тізбекті «блокталған» күйінен «дайын» күйіне ауыстырады.

Block операциясы тізбекті «орындалуда» күйінен «блокталған» күйіне ауыстырады – әдетте, осы операция тізбек қандай да бір оқиғаның болуын тосып тұрған кезде, тізбек үстінен орындалады, мысалы, деректердің енгізілуін немесе қандай да бір ресурстың босауын тоқсан кезде.

Suspend операциясы тізбектің орындалуын тоқтатып қояды.

7.4 Тізбектердің диспетчерленуі және жоспарлануы

Тізбектің диспетчерленумен жоспарлану қағидаларын түсіндіру үшін компьютерде тек бір процессор бар деп санаймыз.

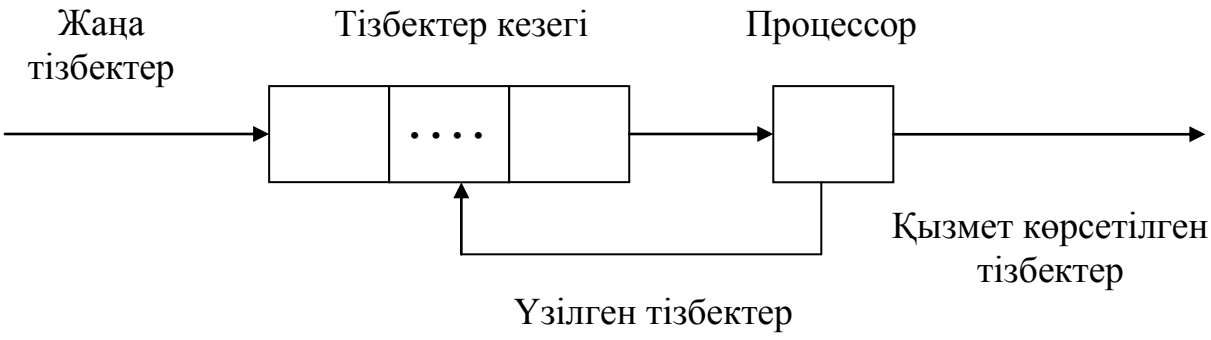
Процессор жұмысының мультибағдарламалық режімін ұйымдастыру үшін оның жұмыс уақыты кванттарға – тізбектерге олардың жұмысы үшін бөлінетін аралықтарға бөлінеді (Windows-та ол шамамен 20 – 30 миллисекунд). Уақыт кванты аяқталған соң тізбектің орындалуы үзіледі де, процессор басқа тізбекке тағайындалады. Тізбектер арасында уақыт кванттарын үлестірумен арнайы бағдарлама - тізбектер менеджері шұғылданады.

Тізбектер менеджері процессорды басқа тізбектің орындалуына ауыстырғанда, ол келесі әрекеттерді орындауға тиіс:

- үзілетін тізбектің контекстін сақтауға;
- іске қосылатын тізбектің контекстін қалпына келтіруге;
- іске қосылатын тізбектің басқаруын тапсыруға.

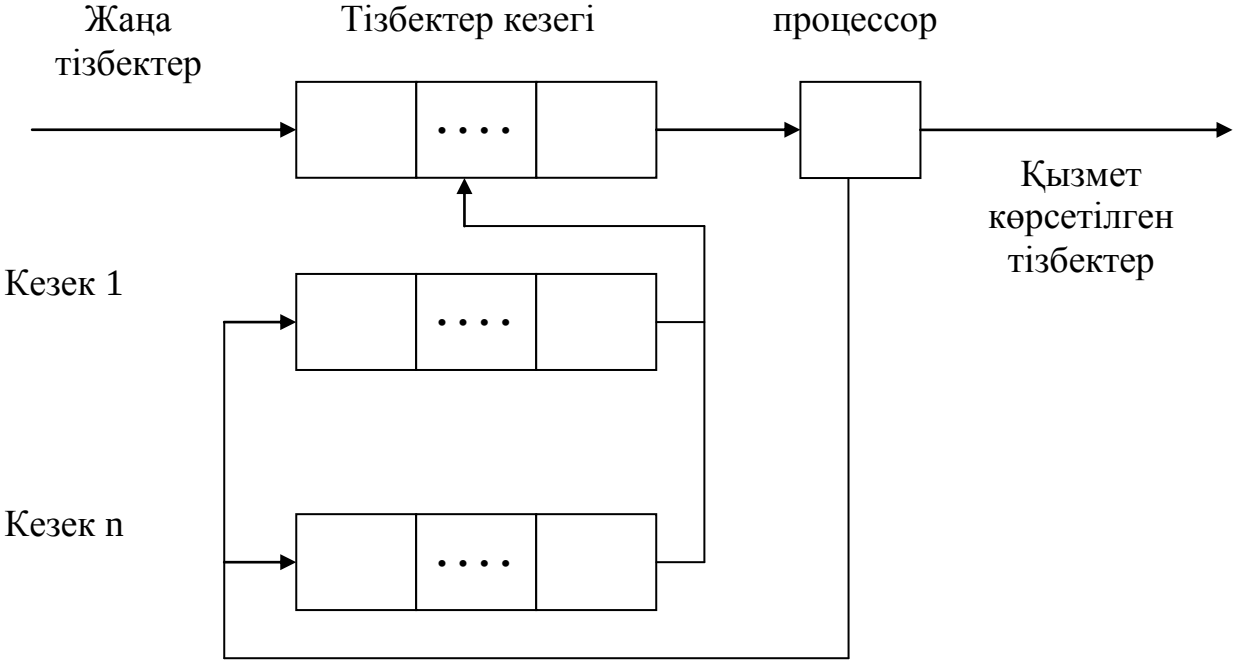
Егерде барлық қызмет көрсетілетін тізбектердің орындау басымдылықтары бірдей болса, онда тізбектерді басқару алгоритмі FIFO (first in – first out) бірінші келіп – бірінші шығасын қағидаты бойынша «кезектің» көмегімен жүзеге асырылады, және үзілген тізбектер кезек соңына тұрады.

Тізбектерге қызмет көрсету сызбалық түрде 7.2-суретте берілген.



7.2-сурет – Тізбектерге қызмет көрсету

Егерде тізбектер әртүрлі басылымдықтарға ие болса, онда басымдылықтары бірдей тізбектер кезектері қалыптастырылады. Бірнеше кезекке қызмет көрсетудің ең қарапайым алгоритмі олардың тізбектерінің басымдылықтарын ескере отырып кезектерге қызмет көрсетуде негізделген. Тізбектерге қызмет көрсету сызбалық түрде 7.3 суретінде көрсетілген.



7.3-сурет – Тізбектерінің басымдылықтары әртүрлі бірнеше кезекке қызмет көрсету

Тізбектерді басқару алгоритмдері жүйенің келесі параметрлерін ескеруге тиіс:

- микропроцессорды жүктеу уақыты максималды болуға тиіс;
- тізбектің жүйеде болу уақыты минималды болуға тиіс;
- тізбектердің күту уақыты минималды болуға тиіс;
- жүйенің өткізгіштік қабілеттілігі максималды болуға тиіс;

- жүйенің өтінімге қызмет көрсету реакциясының уақыты минималды болуға тиіс.

Көп процессорлы компьютерлерде тізбектер менеджері әртүрлі тізбектер кодты әртүрлі процессорларда орындаған кезде тізбектер жұмысының уақытын кванттеуді де, процессорлар жұмысының параллельдігін де пайдаланады.

Уақытты кванттеу қажеттілігі бәрібір қалады, өйткені операциялық жүйе өзінің жеке (жүйелік) тізбектерімен қатар, басқа қосымшалардың тізбектеріне де қызмет көрсетуге тиіс.

7.5 Тізбектерді анықтау

Windows-ғы қосымша бір немесе бірнеше үдерістен тұруы мүмкін. Әр үдеріс бір немесе бірнеше тізбектер тудыруы мүмкін. Әр тізбекке операциялық жүйенің кейбір ресурстары бөлінеді, олар объект түрінде ұсынылады.

Windows-та тізбек дегеніміз операциялық жүйе жүйелік ресурстармен жұмыс істеу үшін процессорлық уақыт бөлетін объект.

Әр тізбекке келесі ресурс тиесілі:

- орындалатын функцияның коды;
- процессор регистрларының жиынтығы;
- қосымшаның жұмысы үшін стек;
- операциялық жүйенің жұмысы үшін стек;
- қауіпсіздік жүйесі үшін ақпараты бар қатынау маркері.

Осы ресурстар тізбектің контекстін құрайды.

Windows жүйесіндегі кез келген объектіге өзінің тіркеу нөмірі - дескрипторы беріледі. Windows-та әр тізбектің дескрипторынан басқа оған өз идентификаторы беріледі. Тізбектердің идентификаторлары қызметтік бағдарламаларымен пайдаланылады және тізбектердің жұмысын бақылауға мүмкіндік береді.

Windows-та тізбектердің екі түрін ажыратады - жүйелік және қолданушы тізбектері.

Жүйелік тізбектерді операциялық жүйенің ядросы іске қосады және олар әртүрлі жүйелік міндеттерді орындау үшін қызмет етеді.

Қолданушы тізбектері қосымшалармен іске қосылады және қолданушының тапсырмаларын шешу үшін қызмет етеді.

.NET технологиясында ресурстарды бөлудің (айрықшалаудың) бұл сызбасы біраз өзгертілген - .NET қосымшасының әр үдерісі қосымшаның бір немесе бірнеше домендерінен тұрады, ал домендердің шегінде бір немесе бірнеше тізбек жұмыс істеуі мүмкін (объект сияқты тізбектің барлық қасиеттері сақталынады).

Әр доменнің ресурстары бір-бірінен оқшаланған. Қосымшаның әртүрлі домендері ешқандай деректерді бірге қолдана алмайды, ол статикалық өрістер немесе тек ортақ доменнің шегінде әртүрлі тізбектермен қолданылуы мүмкін ауқымды айнымалылар болсын олар бірге қолданылмайды.

Сонымен, домен дегеніміз ол үдеріспен тізбек арасында құрастырылатын .NET-гі «оқшаулаудың» немесе деректерді «қорғаудың» қосымша деңгейі.

.NET-гі қандай да бір қосымшалар үшін домен шегінде ***тізбектердің пулдарын*** құруға болады. ***Тізбектердің пулдары*** (ортақтастық) - ағымдағы тізбектер үшін Thread объектісін құру және қосудың орнына ThreadPool.QueueUserWorkItem әдісімен құрастырылатын арнайы шағын кезектер.

Әдетте, тізбек пулының диспетчері пулдағы тізбектер санын автоматты түрде белгілейді (бастапқыда осы мән 50 тең болады). Өздігінен пулдағы тізбектер санының жоғары шегін ThreadPool.SetMaxThreads әдісімен белгілеуге болады.

Жалпы пулдар параллелді жұмыс істеп тұрған қосымшалардың ерікті санының жұмысын ұйымдастыру қажет болғанда пайдаланылады, мысалы, веб-серверлерде. Web Services, ASP.NET, WCF серверлердің және т.б. жұмысында тізбектердің пулдары автоматты түрде бөлінеді.

7.6 Тізбектердің құрылуы

C# тілінде кез келген бағдарлама іске қосылғанда автоматты түрде бағдарламаның басты тізбегі құрылады. Консолдық қосымшада бағдарламаның басты тізбегі Main әдісіне сәйкес келеді.

Қосымша тізбекті іске қосу үшін Thread класының объектісін құру қажет. Тізбек объектісін құрған кезде Thread класының конструкторына ThreadStart әдісі сипатталған делегат беріледі. ThreadStart әдісі делегат әдістің формальді параметрі ретінде аты көрсетілген әдісті қосымша тізбекте іске қосуға мүмкіндік беретін әдіс. Мысалы,

Thread Pervij_dop = new Thread(new ThreadStart(Poick));

мұнда Pervij_dop – бірінші қосымша тізбектің атауы;

Poick –қосымша тізбекте іске қосылатын әдістің атауы.

Жалпы алғанда, делегат «бастапқы» бойынша жұмыс істейді және тізбек объектісін құру келесідей болады:

Thread Pervij_dop = new Thread(Poick);.

Pervij_dop.Start();

Тізбектермен жұмыс істейтін консольдік қосымшаны құрып, System.Threading атаулар кеңестігін қосу қажет.

Thread класында қасиеттер мен әдістер жиынтығы бар, олардың кейбіреулері келесі тізімде көрсетілген:

CurrentThread –қазіргі уақытта орындалып жатқан тізбекке сілтемені қайтаратын, тек оқу үшін арналған статикалық қасиет;

Sleep() – бұл статикалық әдіс, ол ағымдағы тізбектің орындалуын қолданушы белгіленген уақытқа тоқтата тұрады;

IsAlive – тізбектің қосылып тұруына немесе қосылмауына байланысты **true** немесе **false** қайтаратын қарапайым қасиет (тізбек объектісінің құрылуын талап етеді);

IsBackground – тізбектің фондық болуын көрсететін мәнді алу немесе орнату үшін арналған қарапайым әдіс;

Name – тізбектің достық мәтіндік атауын орнату үшін арналған қарапайым әдіс;

Priority – тізбектің басымдылығын алуға/орнатуға мүмкіндік беретін қарапайым әдіс (ThreadPrority тізбелеулеріндегі мәндер қолданылады);

ThreadState – тізбектің күйі жөнінде ақпарат алуға мүмкіндік беретін қарапайым әдіс (ThreadState тізбелеулеріндегі мәндер қолданылады);

Interrupt() – ағымдағы тізбектің жұмысын үзетін қарапайым әдіс;

Join()– басқа тізбектің пайда болуын немесе көрсетілген уақыт аралығында тосатын қарапайым әдіс;

Resume() –тізбектің тоқтатылып тұруынан кейін жұмысын жалғастыратын қарапайым әдіс;

Start() –ThreadStart делегатымен анықталған тізбектің орындалуын бастайтын қарапайым әдіс;

Suspend() – тізбектің орындалуын тоқтатып тұратын қарапайым әдіс. Егерде тізбектің орындалуы тоқтатылып тұрған болса, сонда әдіс ескерілмейді.

Thread класының барлық қасиеттері мен әдістері осы дәрістің қосымшасында келтірілген.

Қосымша тізбек объектісінің құрылуын және оның бағдарламаның басты тізбегінен қосылуының оқу мысалын қарастырамыз. Басты және қосымша тізбектің жұмысын көзбен көріп бақылау үшін оларда дәстүрлі түрде кейбір символдарды немесе цифраларды шығарумен for циклы қолданылады.

Дәстүрлерден алыс кетпейік:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Thread1
    {
        static void WriteCimB()
        {
            for (int i = 1; i <= 100; i++) Console.Write("B");
        }
        static void Main()
        {
            Thread t_dop = new Thread(WriteCimB);
            t_dop.Start();
            for (int i = 1; i <= 100; i++) Console.Write("A");
            Console.ReadLine();
        }
    }
}
```

Работа программы:

AAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

Басты тізбек консольдік терезеге 100 «A» символын шығару үшін арналған, бір мезетте ол сондай-ақ консольдік терезеге «B» деген бір символын 100 рет шығаратын қосымша тізбекті құрады. Бағдарлама жұмысынан олар бір мезетте жұмыс істейтінін көреміз – **for** циклдары тізбектерге бөлінетін уақыт кванттарымен үзіледі. Бір процессорлы компьютерде әр тізбекке жұмыс үшін уақыт кванты белгіленеді (Windows-та ол 20 мс.).

Бағдарламаға келесі жолдарды қосып, қосымша тізбектің күйін зерттеуге болады:

```
Console.WriteLine("состояние доп нити ={0}", t_dop.IsAlive);
Console.WriteLine("приоритет доп нити ={0}", t_dop.Priority);
```

Работа программы:

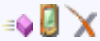
состояние доп нити =True
приоритет доп нити =Normal

AAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB

7 дәрістің қосымшасы

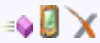
Thread класының әдістерімен қасиеттері

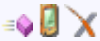
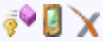
Конструкторы

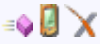
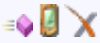
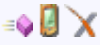
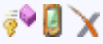
	Имя	Описание
	Thread	Перегружен. Инициализирует новый экземпляр класса Thread.

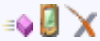
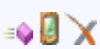
В начало страницы

Методы

	Имя	Описание
	Abort	Перегружен. Вызывает исключение ThreadAbortException в вызвавшем его потоке для того, чтобы начать процесс завершения потока. Вызов данного метода обычно завершает поток.
	AllocateDataSlot	Выделяет неименованную область данных всем потокам. Для улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	AllocateNamedDataSlot	Выделяет именованную область данных всем потокам. Для улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	BeginCriticalRegion	Уведомляет хост, что выполнение близится ко входу к область кода, в которой эффекты прерывания выполнения или неуправляемого выполнения могут повлиять на другие задачи в домене приложения.
	BeginThreadAffinity	Уведомляет хост, что управляемый код близок к выполнению инструкций, зависящих от идентификации текущего потока операционной системы.

	EndCriticalRegion	Уведомляет хост, что выполнение близится ко входу к область кода, в которой эффекты прерывания выполнения или неуправляемой ошибки ограничены текущей задачей.
	EndThreadAffinity	Уведомляет хост об окончании выполнения кодом инструкций, которые зависят от идентификатора текущего потока в операционной системе.
	Equals	Определяет, равен ли заданный объект Object текущему объекту Object. (Унаследовано от Object.)
	Finalize	Освобождает все ресурсы, используемые классом CriticalFinalizerObject. (Унаследовано от CriticalFinalizerObject.) В .NET Compact Framework 3.5 этот член наследуется от Object.Finalize(). В XNA Framework 1.0 этот член наследуется от Object.Finalize().
	FreeNamedDataSlot	Удаляет связь между названием и областью для всех потоков в процессе. Для улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	GetApartmentState	Возвращает значение типа ApartmentState, показывающее состояние апартамента.
	GetCompressedStack	**Устаревшее.** Возвращает объект CompressedStack, который может быть использован для отслеживания стека текущего потока.
	GetData	Извлекает значение из заданной области текущего потока, внутри текущей области текущего потока. Для

		улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	GetDomain	Возвращает текущую область, в которой выполняется текущий поток.
	GetDomainID	Возвращает уникальный идентификатор домена приложения.
	GetHashCode	Возвращает хэш-код текущего потока. (Переопределяет Object.GetHashCode().) В .NET Compact Framework 3.5 этот член наследуется от Object.GetHashCode(). В XNA Framework 1.0 этот член наследуется от Object.GetHashCode().
	GetNamedDataSlot	Ищет именованную область данных. Для улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	GetType	Возвращает объект Type для текущего экземпляра. (Унаследовано от Object.)
	Interrupt	Прерывает работу потока, находящегося в состоянии **WaitSleepJoin**.
	Join	Перегружен. Блокирует вызывающий поток до завершения потока.
	MemberwiseClone	Создает неполную копию текущего объекта Object. (Унаследовано от Object.)
	MemoryBarrier	Синхронизирует доступ к памяти следующим образом: процессор, выполняющий текущий поток, не способен упорядочить инструкции в порядке обращения к памяти, до вызова MemoryBarrier выполняет после того, как память получит доступ после

		вызова MemoryBarrier.
	ResetAbort	Отменяет метод Abort, запрошенный для текущего потока.
	Resume	**Устаревшее.** Возобновляет приостановленную работу потока.
	SetApartmentState	Задает состояние апартамента потока до его запуска.
	SetCompressedStack	**Устаревшее.** Применяет записанное значение CompressedStack к текущему потоку.
	SetData	Задает данные в указанной области для текущей области потока, выполняющегося в данный момент. Для улучшения производительности используйте поля, отмеченные атрибутом ThreadStaticAttribute.
	SetProcessorAffinity	В .NET Compact Framework для Xbox 360, задает сходство процессоров для управляемого потока. Сходство процессоров определяет процессоры, на которых будет выполнен поток.
	Sleep	Перегружен. Блокирует текущий поток на заданное количество миллисекунд.
	SpinWait	Вынуждает поток ожидать количество отсчетов, определенное параметром *iterations*.
	Start	Перегружен. Позволяет планировать выполнение потока.
	Suspend	**Устаревшее.** Приостанавливает работу потока; если работа потока уже приостановлена, не оказывает влияния.
	ToString	Возвращает объект String, который представляет текущий объект Object. (Унаследовано от Object.)

	TrySetApartmentState	Задает состояние апартамента потока до его запуска.
S	VolatileRead	Перегружен. Считывает значение поля. Это значение является последним, записанным каким-либо из процессоров компьютера, независимо от количества процессоров и от состояния кэш-буфера процессоров.
S	VolatileWrite	Перегружен. Записывает значение непосредственно в поле, так что оно становится видимым для всех процессоров компьютера.

В начало страницы

Свойства

	Имя	Описание
	ApartmentState	**Устаревшее.** Получает или задает состояние апартамента для данного потока.
S	CurrentContext	Возвращает текущий контекст, в котором выполняется поток.
	CurrentCulture	Получает или задает язык и региональные параметры для текущего потока.
S	CurrentPrincipal	Получает или задает текущего участника потока (для безопасности на основе ролей).
S	CurrentThread	Возвращает выполняющийся в данный момент поток.
	CurrentUICulture	Получает или задает текущие язык и региональные параметры, используемые диспетчером ресурсов для поиска ресурсов, связанных с языком и региональными параметрами, во время выполнения.
	ExecutionContext	Возвращает объект ExecutionContext,

		содержащий сведения о различных контекстах текущего потока.
	IsAlive	Возвращает значение, показывающее статус выполнения текущего потока.
	IsBackground	Получает или задает значение, показывающее, является ли поток фоновым.
	IsThreadPoolThread	Возвращает значение, показывающее, принадлежит ли поток к группе управляемых потоков.
	ManagedThreadId	Возвращает уникальный идентификатор текущего управляемого потока.
	Name	Получает или задает имя потока.
	Priority	Получает или задает значение, указывающее на планируемый приоритет потока.
	ThreadState	Возвращает значение, содержащее состояния текущего потока.

В начало страницы