Дәріс 4. Monitor класы. Мьютекс пен семафорды пайдалану.

Дәрістің мақсаты: Студенттерде ағындарды синхронизациялау құралдарының қызметі және жұмыс істеу ерекшеліктері туралы түсінік қалыптастыру.

Дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Monitor класының қызметін түсіну;
- Мьютекс пен семафордың қызметтерін түсіну.

Monitor класы және бұғаттау

lock кілт сөзі C# бағдарламасында System.Threading аттар кеңістігінде орналасқан Monitor сыныбында анықталған үндестіру құралдарына жылдам қатынау тәсілі болып табылады. Бұл сыныпта, атап айтқанда, бірқатар әдістер айқындалған үндестіруді басқару үшін. Мысалы, нысанды құрсаулауды алу үшін Enter() әдісі, ал құрсаулауды алу үшін Exit() әдісі шақырылады. Төменде осы әдістердің жалпы нысандары келтірілген:

public static void Enter(object obj)

public static void Exit(object obj)

мұнда obj үндестірілетін нысанды білдіреді. Егер нысан қол жетімді болмаса, онда Enter() әдісін шақырғаннан кейін шақырушы ағын нысан қол жетімді болғанша күтеді. Дегенмен, Enter() және Exit() әдістері сирек қолданылады, себебі lock операторы автоматты түрде ағынды үндестірудің баламалы құралдарын ұсынады. Сол себепті lock операторы C# бағдарламасында бағдарламалау кезінде нысанды құрсаулауды алу үшін "анағұрлым артықшылықты" болып табылады.

Дегенмен, Monitor класының бір әдісі пайдалы болуы мүмкін. Бұл жалпы нысандарының бірі төменде келтірілген TryEnter() әдісі.

public static bool TryEnter(object obj)

Бұл әдіс, егер шақырушы ағын obj нысаны үшін құрсаулау алса, немесе ол false логикалық мәнін қайтарса, логикалық мәнді қайтарады.

Бірақ кез келген жағдайда шақырушы ағым өз кезегін күтуге тура келеді. TryEnter() әдісінің көмегімен егер талап етілетін нысан уақытша қол жетімді болмаса, ағынды үндестірудің баламалы нұсқасын іске асыруға болады.

Wait(), Pulse() және PulseAll() әдістерінің көмегімен ағындар арасындағы хабар

Келесі жағдайды қарастырайық. T ағыны lock кодтық блогында орындалады және ол уақытша қол жетімді емес R ресурсына қол жеткізуді талап етеді. Сонда T ағынына не істеу керек? Егер T ағыны R ресурсының босатылуын күтіп, белгілі бір нысанда ұйымдастырылған сауалнама цикліне кіретін болса, онда ол сол арқылы оған басқа ағындардың қол жеткізуіне тосқауыл қоя отырып, тиісті объектіні байланыстырады. Бұл ең оңтайлы шешім емес, себебі ол көп ағынды орта үшін бағдарламалаудың артықшылықтарынан жартылай айырады. Неғұрлым жетілдірілген шешім объектіні уақытша босатып, сол арқылы басқа да ағындардың орындалуына мүмкіндік беру болып табылады. Мұндай тәсіл ағындар арасындағы хабарламаның кейбір түріне негізделеді, соның арқасында бір ағын екіншісіне оның бұғатталғаны және басқа ағынның өзінің

орындалуын қайта бастауы мүмкін екендігі туралы хабарлай алады. Ағындар арасындағы хабарлама C# -де Wait(), Pulse() және PulseAll() әдістерінің көмегімен ұйымдастырылады.

Wait(), Pulse() және PulseAll() әдістері Monitor класында анықталған және тек блоктың құрсауланған элементінен ғана шақырылуы мүмкін. Олар былайша қолданылады. Ағынды орындау уақытша құрсауланғанда, ол Wait() әдісін шақырады. Нәтижесінде ағын күту күйіне өтеді, ал тиісті нысаннан бұғаттау алынады, бұл нысанды басқа ағымда пайдалануға мүмкіндік береді.

Бұдан әрі күтуші ағын басқа ағын бұғаттаудың ұқсас күйіне кіргенде белсендіріледі және Pulse() немесе PulseAll() әдісін тудырады. Pulse() әдісін шақыру кезінде бұғаттауды алуға өз кезегін күтетін бірінші ағынды орындау қайта басталады. Ал PulseAll() әдісін шақыру барлық күтетін ағындарға бұғаттауды алып тастау туралы белгі береді.

Төменде Wait () әдісінің жиі қолданылатын екі түрі көрсетілген.

public static bool Wait(object obj)

public static bool Wait (object obj, int миллисекунд _ тұрып қалу)

Бірінші нысанда күту объектіні босату туралы хабарламаға дейін, ал екінші нысанда - объектіні босату туралы хабарламаға дейін, сондай-ақ тұрып қалу миллисекунд _ санын көрсететін уақыт кезеңі аяқталғанға дейін созылады. obj екі пішінінде де босатылуы күтілетін нысанды білдіреді.

Төменде Pulse() және PulseAll() әдістерінің жалпы нысандары келтірілген:

public static void Pulse(object obj)

public static void PulseAll(object obj)

мұнда obj бос нысанды білдіреді.

Егер Wait(), Pulse() және PulseAll() әдістері үндестірілген кодтың сыртында, мысалы, lock блогынан, онда SynchronizationLockException шығарылады.

Мысал 1. Monitor класын бұғаттау барысында пайдалану.

```
using System;
using System.Threading;
class TickTock {
object lockOn = new object();  // бұғаттауға қажетті жабық объект
public void Tick(bool running) {
lock(lockOn) { // әдісті бұғаттау
if(!running) { // сағатты тоқтату
Monitor.Pulse(lockOn); // күтудегі ағындарға хабарлау
return;
}
Console.Write("тик ");
Monitor.Pulse(lockOn); // Tock() әдісінің орындалуына рұқсат беру
Monitor.Wait(lockOn); // Tock() әдісінің аяқталуын күту
}
}
public void Tock(bool running) {
lock(lockOn) { // әдісті бұғаттау
if(!running) { // сағатты тоқтату
```

```
Monitor.Pulse(lockOn); // күтудегі ағындарға хабарлау
return;
}
Console.WriteLine("так");
Monitor.Pulse(lockOn); // Tick()әдісінің орындалуына рұқсат беру
Monitor.Wait(lockOn); // Tick()әдісінің аяқталуын күту
}
}
}
class MyThread {
public Thread Thrd;
TickTock ttOb;
// Жаңа ағын құру
public MyThread(string name, TickTock tt) {
Thrd = new Thread(this.Run);
ttOb = tt;
Thrd.Name = name;
Thrd.Start();
}
// Жаңа ағын жұмысын бастау.
void Run() {
if(Thrd.Name == "Tick") {
for(int i=0; i<5; i++) ttOb.Tick(true);
ttOb.Tick(false);
}
else {
for(int i=0; i<5; i++) ttOb.Tock(true);
ttOb.Tock(false);
}
}
}
class TickingClock {
static void Main() {
TickTock tt = new TickTock();
MyThread mt1 = new MyThread("Tick", tt);
MyThread mt2 = new MyThread("Tock", tt);
mt1.Thrd.Join();
mt2.Thrd.Join();
Console.WriteLine("Сағат тоқтатылды");
}
}
```

Мьютексті пайдалану

Мьютекс үндестіруді болдырмайтын нысанды білдіреді. Бұл ағынмен тек кезекпен ғана алынуы мүмкін дегенді білдіреді. Мьютекс ортақ ресурс бір мезгілде бір ғана ағында пайдаланылуы мүмкін жағдайларға арналған. Жүйелік журнал бірнеше процестерде бірлесіп пайдаланылады, бірақ олардың біреуінде ғана деректер осы журналдың файлына кез келген уақытта жазылуы мүмкін. Бұл жағдайда процестерді үндестіру үшін мьютекс өте қолайлы.

Мьютекс System.Threading.Mutex. Оның бірнеше құрастырушысы бар. Төменде ең көп қолданылатын екі конструктор келтірілген.

public Mutex()

public Mutex(bool initiallyOwned)

Мысал 2. Мьютекс көмегімен бұғаттау.

```
using System;
using System.Threading;
class SharedRes {
public static int Count = 0; // ортақ қолданылатын ресурс
public static Mutex Mtx = new Mutex(); // ортақ қолданылатын ресурсқа қол жеткізуді
                                       // басқаратын мьютекс
}
// Бұл ағында SharedRes.Count айнымалысы инкременттеледі
class IncThread {
int num;
public Thread Thrd;
public IncThread(string name, int n) {
Thrd = new Thread(this.Run);
num = n;
Thrd.Name = name;
Thrd.Start();
}
// ағынға кіру нүктесі.
void Run() {
Console.WriteLine(Thrd.Name + " мьютексті күтуде.");
// Получить мьютекс.
SharedRes.Mtx.WaitOne();
Console.WriteLine(Thrd.Name + " мьютексті қабылдады.");
do {
Thread.Sleep(500);
SharedRes.Count++;
Console.WriteLine(Thrd.Name + " ағынында, SharedRes.Count = " + SharedRes.Count);
num--;
} while(num > 0);
Console.WriteLine(Thrd.Name + " мьютексті босатты.");
// Мьютексті босату.
SharedRes.Mtx.ReleaseMutex();
}
}
//Бұл ағында SharedRes.Count айнымалысы декременттеледі.
class DecThread {
int num;
public Thread Thrd;
public DecThread(string name, int n) {
Thrd = new Thread(new ThreadStart(this.Run));
num = n;
Thrd.Name = name;
Thrd.Start();
}
// ағынға кіру нүктесі.
void Run() {
Console.WriteLine(Thrd.Name + " мьютексті күтуде.");
```

```
// Получить мьютекс.
SharedRes.Mtx.WaitOne();
Console.WriteLine(Thrd.Name + " мьютексті қабылдады.");
do {
Thread.Sleep(500);
SharedRes.Count--;
Console.WriteLine(Thrd.Name + " ағынында, SharedRes.Count = " + SharedRes.Count);
num--;
} while(num > 0);
Console.WriteLine(Thrd.Name + " мьютексті босатты.");
// Мьютексті босату.
SharedRes.Mtx.ReleaseMutex();
}
}
class MutexDemo {
static void Main() {
// Екі ағынды құру
IncThread mt1 = new IncThread("Инкременттеуші ағын", 5);
Thread.Sleep(1); // ағынды іске қосу
DecThread mt2 = new DecThread("Декременттеуші ағын", 5);
mt1.Thrd.Join();
mt2.Thrd.Join();
}
}
```

Семафорды пайдалану

Семафор мьютекске ұқсас, жалпы ресурстарға бір емес, бірнеше ағындарға бір мезгілде рұқсат береді. Сондықтан семафор бірқатар ресурстарды үндестіруге жарамды. Семафор осы мақсат үшін есептегішті пайдалана отырып, ортақ ресурстың қатынасын басқарады. Егер санағыштың мәні нөлден үлкен болса, онда ресурсқа қатынасуға рұқсат етіледі. Ал егер бұл мән нөлге тең болса, онда ресурсқа қол жеткізуге тыйым салынады. Есептегіштің көмегімен рұқсаттардың саны есептеледі. Демек, ресурстарға қол жеткізу үшін ағын семафордан рұқсат алуы тиіс.

Мысал 3. Семафор көмегімен бұғаттау.

```
using System;
using System.Threading;
// Келесі ағын өзінің тек екі экземплярын қатар орындауға рұқсат береді
class MyThread {
public Thread Thrd;
// Семафор құрылады, ол бастапқыда бар 2 рұқсаттың екеуін береді
static Semaphore sem = new Semaphore(2, 2);
public MyThread(string name) {
Thrd = new Thread(this.Run);
Thrd.Name = name;
Thrd.Start();
}
// Ағынға кіру нүктесі.
void Run() {
Console.WriteLine(Thrd.Name + " рұқсатты күтуде.");
```

```
sem.WaitOne();
Console.WriteLine(Thrd.Name + " рұқсатты алды.");
for(char ch='A'; ch < 'D'; ch++) {
Console.WriteLine(Thrd.Name + " : " + ch + " ");
Thread.Sleep(500);
}
Console.WriteLine(Thrd.Name + " рұқсатты босатты.");
// Семафорды босату.
sem.Release();
}
}
class SemaphoreDemo {
static void Main() {
// Үш ағын құру.
MyThread mt1 = new MyThread("#1 ағын");
MyThread mt2 = new MyThread("#2 ағын");
MyThread mt3 = new MyThread("#3 ағын");
mt1.Thrd.Join();
mt2.Thrd.Join();
mt3.Thrd.Join();
}
}
```