

Визуал програмчлал

Ү.ИТ203

Лекц-12-13

Гарчиг

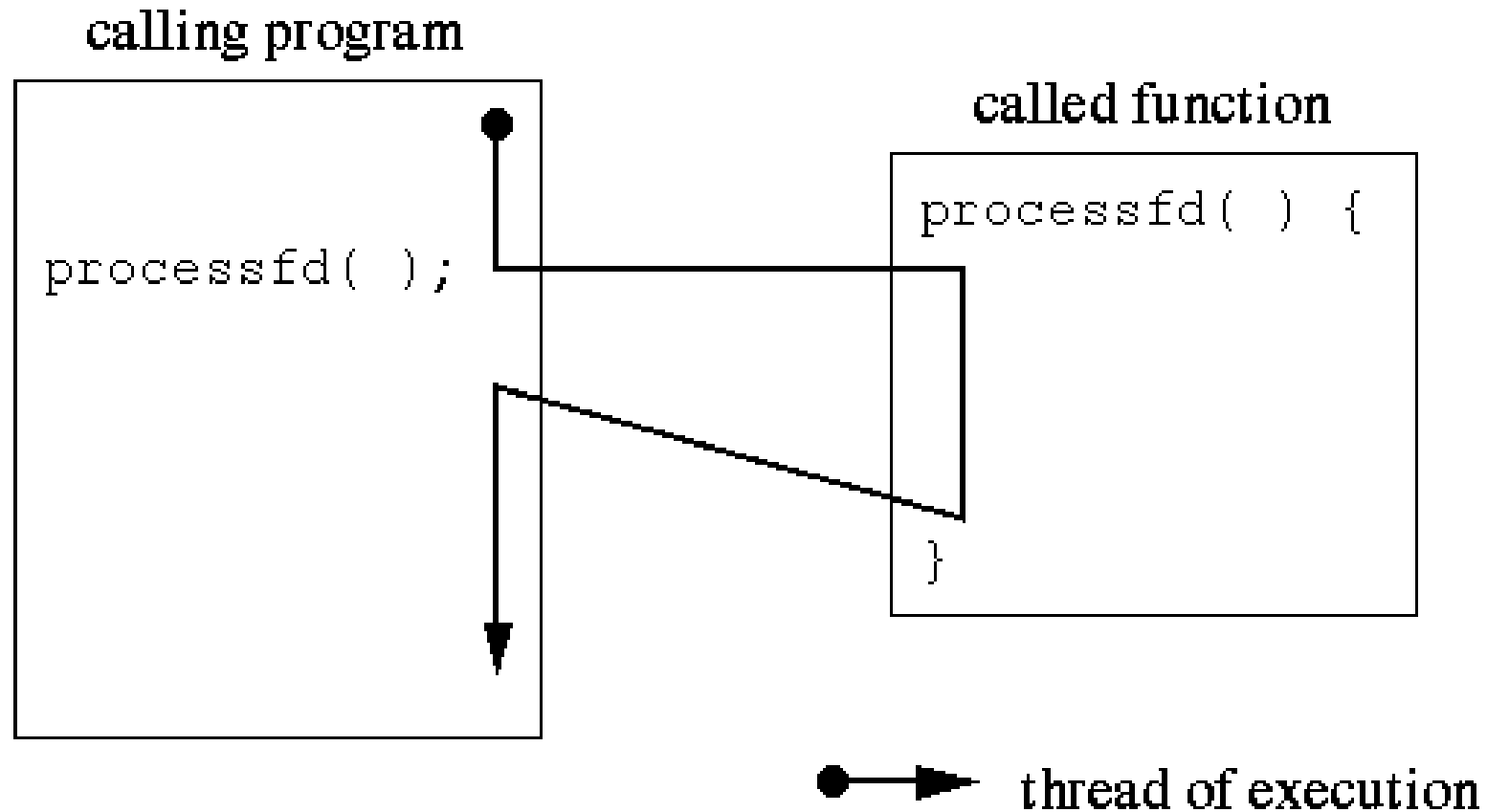
- Програмчлалын хоёр хувилбар
- Thread() / Stream
- Жишээ

Урсгал

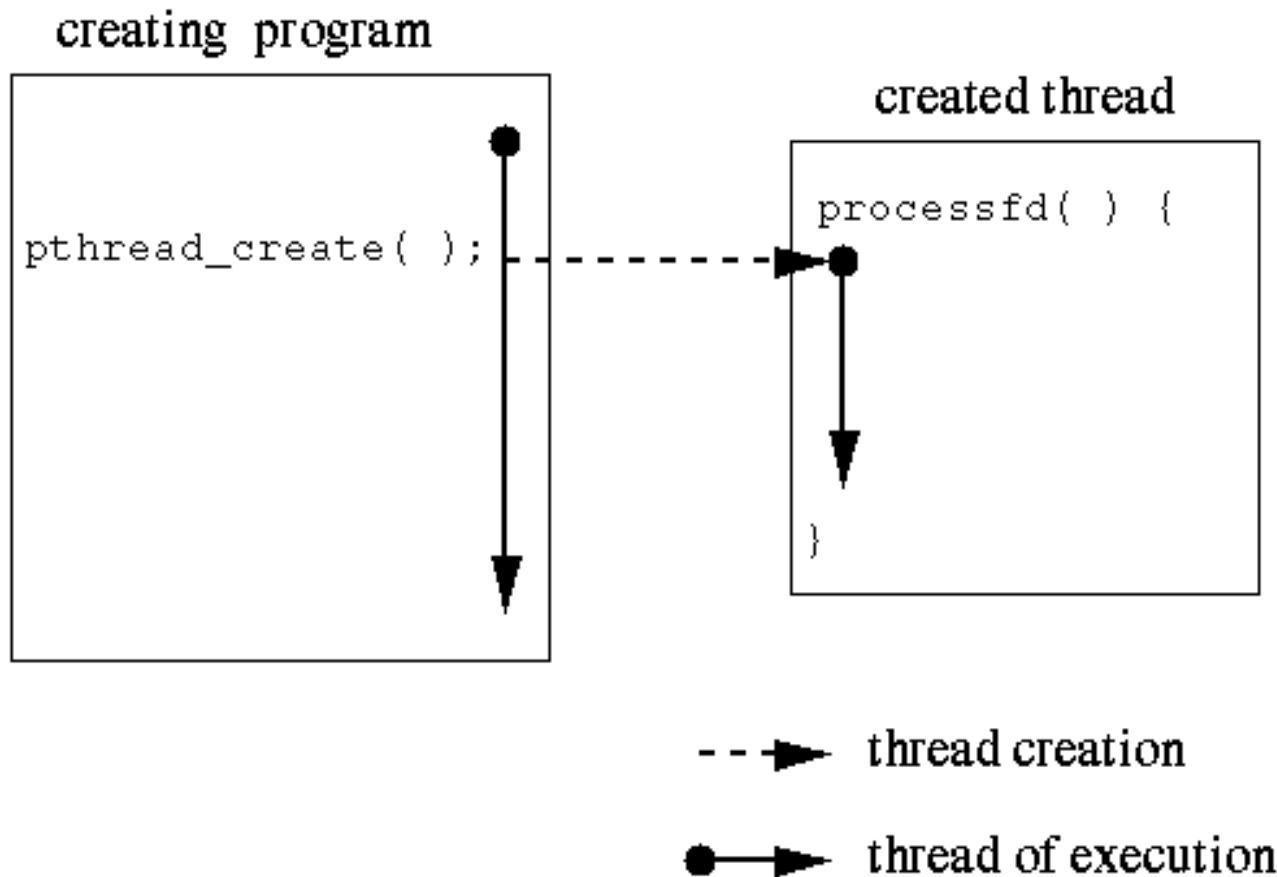
Програм нь дараах 2 аргын аль нэгээр ажиллахаар зохиомжлогдоно.

- Синхрон-Шугаман
- Асинхрон-Параллель

Нэг урсгалтай програмын жишээ



Олон урсгалтай програмын жишээ

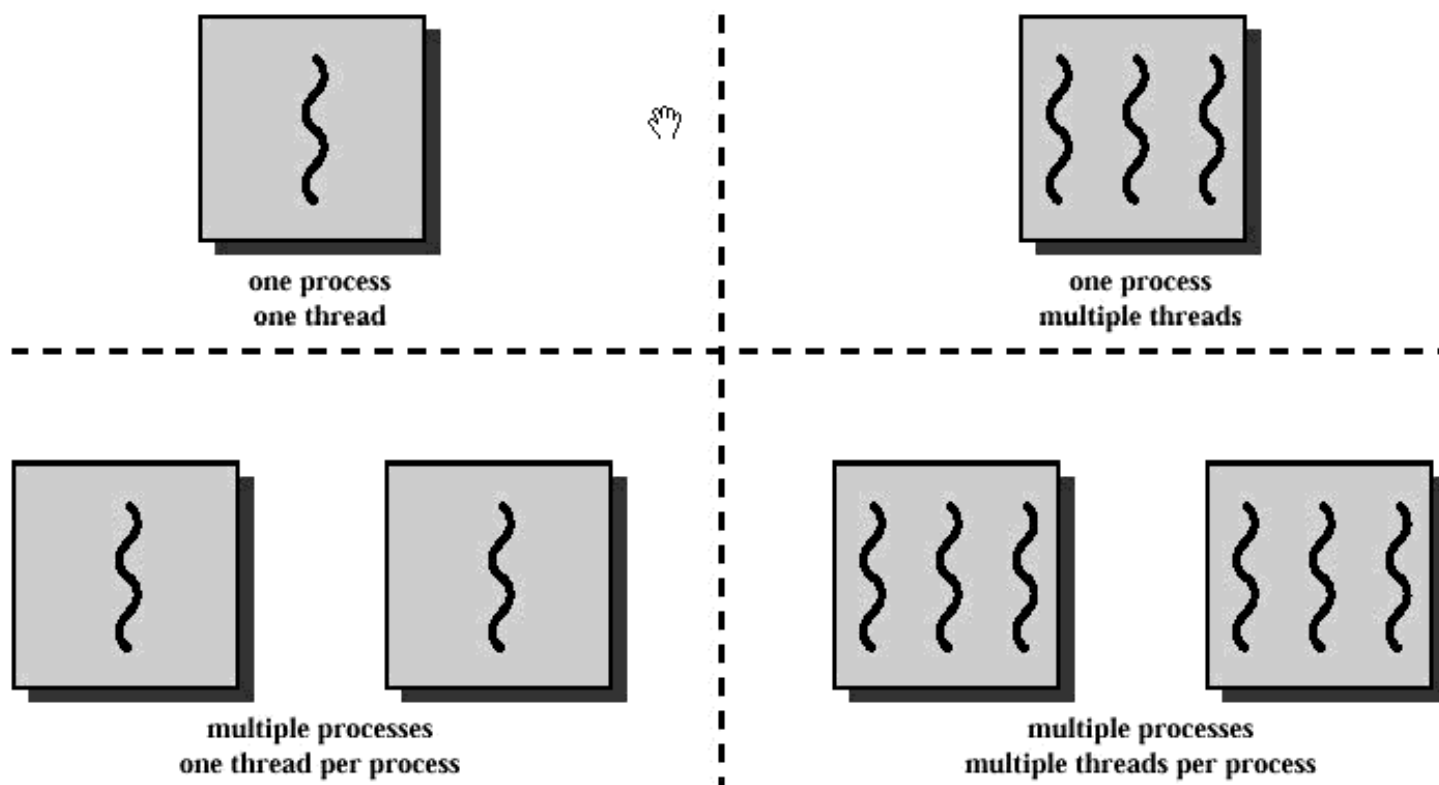


Stream(Урсгал)

Ийнхүү олон урсгалт гэдэг нь зэрэгцээ бодолтын тусгай хэлбэр юм. Тэгэхдээ зэрэгцээ бодолтын хоёр янзын хэлбэр оршин байдаг.

- Үүний нэг нь процесс дээр
- нөгөө нь урсгал дээр үндэслэгдсэн байдаг.

Урсгал болон процессын загварууд



Урсгал

- Уламжлалт нэг урсгалт програм нэг бодлого дуусах хүртэл дараачийн бодлого хүлээж байгаад дуусмагц ажиллаж эхэлдэг. Ийм бодлогод CPU ихээхэн цаг зарцуулдаг.
- Олон урсгалт арга нь энэ хугацааг дээд зэргээр ашиглаж Олон даалгаврыг зэрэг гүйцэтгэх боломж олгоно.

Урсгал

- C# нь олон урсгалт програмчлалын арга дээр тулгуурладаг. Олон урсгалт програм нь нэг зэрэг биелэх, Харилцан бие биетэйгээ өрсөлдөх хоёр буюу түүнээс олон хэсгийг агуулна.
- Тийм програмын хэсэг бүрийг урсгал гэж нэрлэх бөгөөд урсгал бүр нь өөр өөрөөр биелэх замыг тодорхойлдог.
- Асинхрон програмчлалын гол түлхүүр нь **thread** ашиглах юм.

Thread гэж юу вэ?

- Thread гэдэг нь бие даан зэрэгцэн ажиллах ХЭСЭГ КОД ЮМ.
- Thread нь програмд олон даалгаврыг зэрэг гүйцэтгэх боломж олгодог.

Thread гэж юу вэ?

Thread онцлог

- Програмын хурд бүтээмжийг нэмэгдүүлнэ. Жишээ нь нэг **Thread** нь хэрэглэгчээс өгөгдөл оруулахыг хүлээж, нөгөө нь хэвлэлт хийж байж болно.
- .NET – ээр биш ҮС-ээр удирдагдаж ажилладаг. .NET нь гүүр болох бүхий классуудаар хангагдсан байдаг.

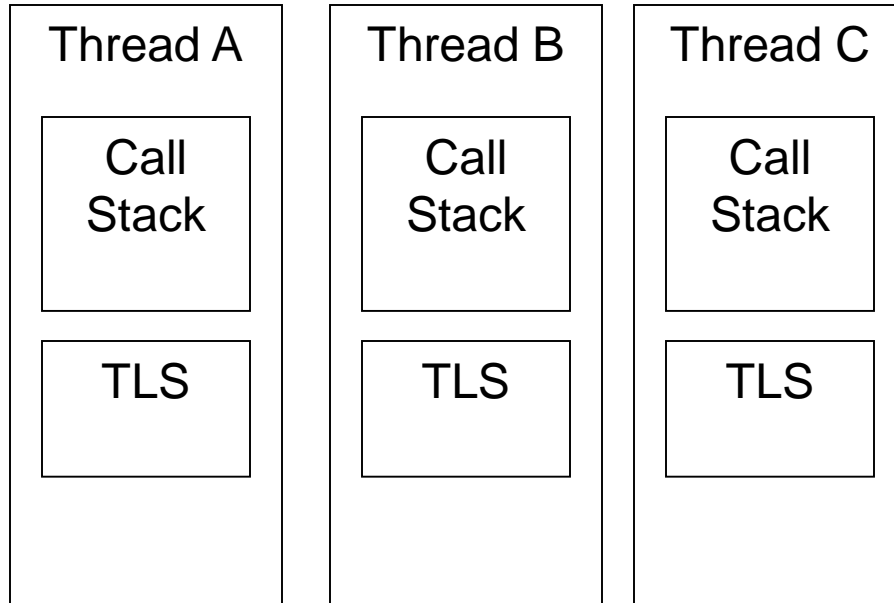
Thread гэж юу вэ?

- Програмыг ачааллахад програмын эхний цэг болох үндсэн **Thread** ажиллаж эхэлдэг. Энэ нь бол `main()` метод юм.
- .NET нь физик Thread үүсгэхгүй, энэ нь үйлдлийн системийн үүрэг юм. Харин физик Thread-ийг орлон ажиллах Thread классаар хангадаг.

Програм-процесс-трэд

Process

App.EXE



Олон трэд

Хэрэв трэд энэ үед ажлаа дуусгаж амжаагүй бол түүний төлвийн мэдээллийг хадгалж авах ба энэ мэдээллийг дараагийн идэвхжилтэнд ашигладаг. Төлвийн мэдээлэлд регистэрүүд, стекийн заагч, програмын тоолуур зэрэг мэдээллүүд багтана. Эдгээр бүх мэдээллүүд нь TLS гэж нэрлэгдэх санах ойн мужид хадгалагдана.

Олон трэд

- Нэг CPU бүхий системийн хувьд нэг эгшинд зөвхөн ганц трэд ажиллаж чадна. Ямар трэд эхэлж ажиллах нь тэдгээрийн зэрэглэлээс хамаарна.
- Трэд дарааллын эхэнд ирмэгц түүн доторх кодын цуваа тодорхой хугацааны завсарт ажиллаад дараагийн трэдэд удирдлага шилждэг. Үүнийг хугацаагаар хуваах арга гэж хэлнэ.

Трэд зэрэглэл

- Трэдийн ажиллах дараалал зэрэглэлээс хамаарна. Хэрэв нэг трэд ажиллаж байхад өөр нэг өндөр зэрэглэлтэй трэд ажиллахад болвол түүнд зам тавьж өгнө.
- Хэрэв хэд хэдэн адил зэрэглэлтэй трэдүүд байвал үйлдлийн систем тэгш хувиарлах(round-robin) аргыг ашиглана.

Трэд зэрэглэл

Трэдийн Priority пропериг ашиглан зэрэглэлийг тогтоож болно. Үүнд

- Lowest,
- BelowNormal,
- Normal,
- AboveNormal
- Highest

Трэд зэрэглэл

Трэдийн зэрэглэлийг өөрчилснөөр хүсээгүй үр дүнд гарч болох учраас болгоомжтой хандах хэрэгтэй.

Үүнд:

- Трэд хэдийгээр хамгийн өндөр зэрэглэлтэй байсан ч бусад трэдээр блоклогдож болно.
- Трэдийг зэрэглэлийг өсгөснөөр түүнийг үйлдлийн системийн бусад трэдүүдтэй урцалдахад хүргэх ба энэ нь эргээд системийн гүйцэтгэлд сөрөг нөлөө үзүүлдэг.

Трэд төрөл

.NET – д ажиллах онцлогоос нь хамаарч трэдүүдийг дараах 2 төрлөөр ангилдаг.

- Ил - Шинэ трэд ил байдлаар үүснэ.
- Далд - `IsBackground = true;`

Трэдийн төлөв

Трэд амьдрах хугацаандаа хэд хэдэн төлөвт шилжиж байдаг. Эхлээд `unstarted` төлөвт эхлэнэ. Дараа нь CPU ажиллуулж эхлэхэд `Running` төлөвт орно. Хэрэв түүнд оногдсон хугацаа дуусвал `ҮС` түүн дээр түр зогсох `ХЭМЖЭЭ` авдаг.

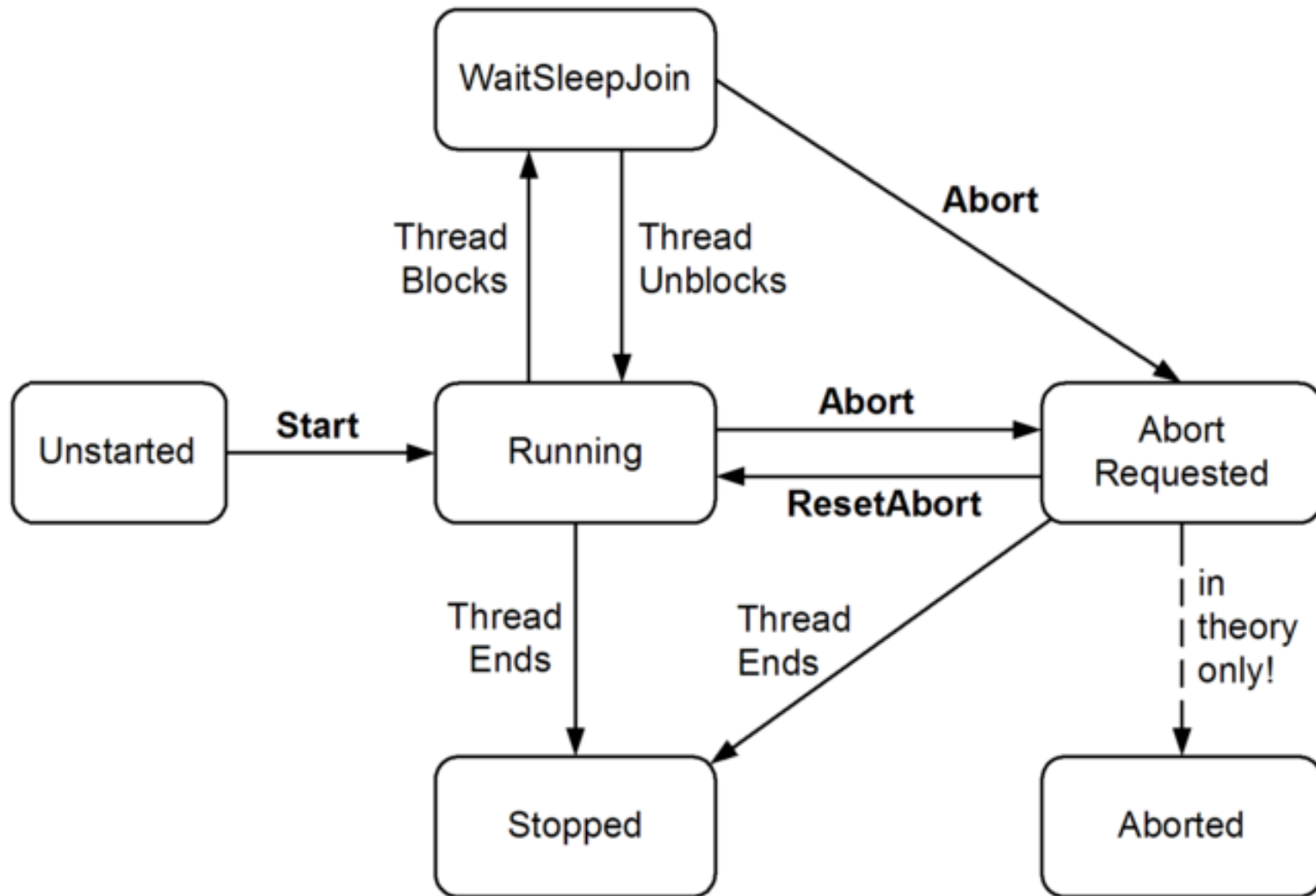
Трэдийн төлөв

Хэрэв ажиллуулж дууссан бол Stopped төлөвт орно. Өөр нэг waitsleepjoin нь трэд нөөц сулархыг хүлээх эсвэл өөр нэг трэд дуусахыг хүлээх төлөв юм. Энэ төлөв дуусахад трэд running төлөвт орно.

Трэдийн төлөв

| Төлөв | Тайлбар |
|------------------|---|
| Aborted | Замыг тасалсан |
| AbortRequested | Өөр замаас тухайн замыг таслах хүсэлт ирсэн |
| Background | Зам нь ард ажиллах зам (background thread) байдлаар ажиллаж байна |
| Running | Зам ажиллаж байна |
| Stopped | Замын ажиллагаа зогссон |
| StopRequested | Зам зогсох хүсэлт илгээсэн |
| Suspended | Зам түр зогссон |
| SuspendRequested | Түр зогсоох хүсэлт илгээсэн |
| Unstarted | Зам үүссэн, гэхдээ эхлээгүй |
| WaitSleepJoin | Зам түгжигдсэн эсвэл түр зогссон эсвэл өөр зам холбогдохыг хүлээж байна |

Трэдийн төлөв



Урсгалын төлөв

Урсгалд хэд хэдэн төлөв бий.

- Биелэх:үндсэн үүргээ гүйцэтгэж байгаа төлөв
- Биелэхэд бэлэн: дөнгөж CPU-аас ажиллах цаг зөвшөөрөл авсан төлөв
- Зогсоогдсон: биелэж байсан урсгалыг зогсоож түүнд үйлчилэх хугацааг зогсоох

Урсгалын төлөв

- Үргэлжлэх: Зогсоогдсон урсгалыг зогссон газар тухайн үеийн төлөвөөс нь эхлэн үргэлжлүүлэх төлөв
- Хүлээх-хаах: урсгал системийн болон бусад нөөц хүлээх хаах төлөв
- Төгсөх: биелэж байгаа урсгалын ажиллагааг зогсоох. Төгссөн урсгалыг дахин үргэлжилүүлж болохгүй.

Урсгалуудын зиндаа

- Урсгал бүр дээр тухайн урсгалыг боловсруулах эрэмбэ дарааллыг бусад урсгалуудтай харьцуулан түүний зиндааг тогтоодог. Урсгалын зиндаа нь нэг урсгалын зиндааг нөгөө урсгалын зиндаатай харьцуулсан бүхэл тоо юм. Өндөр зиндааны урсгал доод зиндааныхаас хурдан биелдэггүй.

Урсгалуудын зиндаа

Урсгалын зиндааг нэг биелэж байгаа урсгалаас дараачийн урсгалд шилжин орохыг шийдэхэд ашигладаг. Энэнийг шилжилтийн контекст гэж нэрлэдэг.

Дараах дүрмээр шилжилтийн контекстийг тодорхойлдог.

- Урсгал нь сайн дураараа татгалзаж болно.
- Урсгалыг илүү зиндааны урсгалаар түдгэлзүүлж болно.

Гол урсгал

Програмыг ажиллуулах үед нэг урсгал биелэж эхлэн бөгөөд тэр нь удаан үргэлжлэхгүй. Түүнийг програмын эхнээс биелэж эхлэх учраас програмын гол урсгал гэж хэлдэг. Гол урсгал чухал болох нь дараах шалтгаантай

- Бусад бүх охин урсгалууд төрнө
- Сүүлчийн урсгал байна.

Гол урсгал

Гол урсгал нь програмыг ажиллуулахад автоматаар үүсэх бөгөөд тэр нь Thread объектоор удирдагдаж болно. Удирдлагыг зохион байгуулахын тулд static гишүүн CurrentThread() дүрмийн хувилбарыг авна. Энэ дүрэм дуудагдсан үедээ урсгалын хувилбарыг буцаана. Тэгвэл бид гол урсгалын хувилбарыг буцаана.

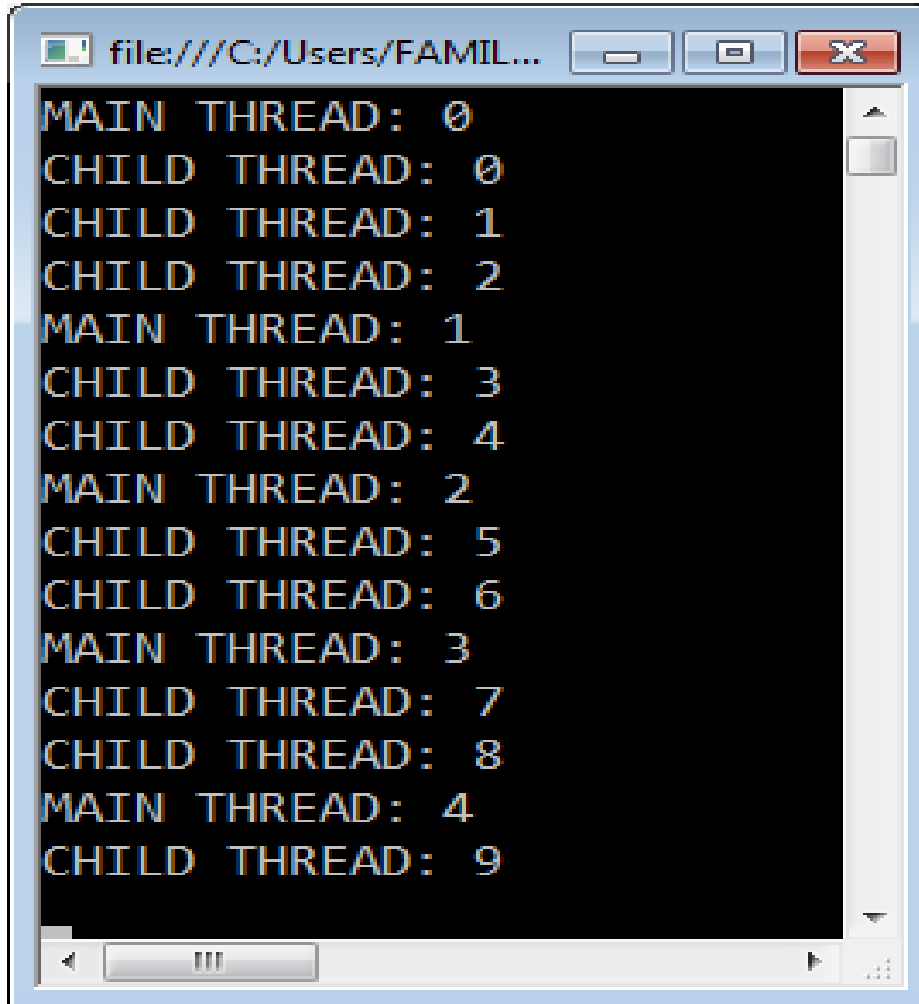
Жишээ-1

```
static void Main()  
{  
    ThreadStart job = new ThreadStart(ThreadJob);  
    Thread thread = new Thread(job);  
    thread.Start();  
  
    for (int i = 0; i < 5; i++)  
    {  
        Console.WriteLine("MAIN THREAD: {0}", i);  
        Thread.Sleep(1000);  
    }  
    Console.ReadLine();  
}
```

Жишээ-2

```
static void ThreadJob()  
{  
    for (int i = 0; i < 10; i++)  
    {  
        Console.WriteLine("CHILD  
THREAD: {0}", i);  
        Thread.Sleep(500);  
    }  
}
```

Үр дүн



A screenshot of a Windows command prompt window. The title bar shows the file path "file:///C:/Users/FAMIL...". The window contains a list of thread identifiers, alternating between "MAIN THREAD" and "CHILD THREAD", each followed by a number from 0 to 9. The text is displayed in a monospaced font with a yellowish-green color on a black background. The window has standard Windows controls (minimize, maximize, close) and a scrollbar on the right side.

```
file:///C:/Users/FAMIL...  
MAIN THREAD: 0  
CHILD THREAD: 0  
CHILD THREAD: 1  
CHILD THREAD: 2  
MAIN THREAD: 1  
CHILD THREAD: 3  
CHILD THREAD: 4  
MAIN THREAD: 2  
CHILD THREAD: 5  
CHILD THREAD: 6  
MAIN THREAD: 3  
CHILD THREAD: 7  
CHILD THREAD: 8  
MAIN THREAD: 4  
CHILD THREAD: 9
```


Thread ангийн зарим дүрмүүд

| Дүрмүүд | Агуулга |
|-----------|--|
| isAlive() | Өөр урсгал биелэгдэх эсэхийг тодорхойлох |
| Join() | Урсгал төгсөхийг хүлээх |
| Sleep() | Тодорхой хугацаанд урсгалыг зогсоох |
| Start() | Урсгалыг оруулах |

isAlive()

Гол урсгал хамгийн сүүлд төгсөх ёстой. Өөрөөр хэлбэл ерөнхий урсгал дотор нилээд удаашруулан барьж охин урсгал төгссөний дараа өөрөө төгсдөг байх. Thread ангид урсгал төгссөн эсэхийг тодорхойлох арга байдаг. Энэ нь isAlive() дүрэм ашиглах юм. Энэ дүрэм нь хэрэв урсгал дуудагдсан, биелэж байх үед true утга буцаана.

Жишээ-3

```
using System;
using System.Threading;
public class ThreadState
{
    static void WorkerFunction()
    {
        for (int i = 1; i < 10; i++)
        {
            Thread.Sleep(2000);
            Console.WriteLine("Worker: " +
                Thread.CurrentThread.ThreadState);
        }
    }
}
```

Жишээ-3

```
static void Main()
{
    //string ThreadState;
    Thread t = new Thread(new ThreadStart(WorkerFunction));

    t.Start();
    while (t.IsAlive)
    {
        Console.WriteLine("Still waiting. I'm going back to sleep.");
        Thread.Sleep(1000);
    }
    Console.WriteLine(t.ThreadState);
    Console.Read();
}
```

Still waiting. I'm going back to sleep.

Still waiting. I'm going back to sleep.

Worker: Running

Still waiting. I'm going back to sleep.

Still waiting. I'm going back to sleep.

Worker: Running

Still waiting. I'm going back to sleep.

Still waiting. I'm going back to sleep.

Worker: Running

Still waiting. I'm going back to sleep.

Still waiting. I'm going back to sleep.

Worker: Running

Still waiting. I'm going back to sleep.

Still waiting. I'm going back to sleep.

Worker: Running

Stopped