

```

// R_FatNumForName ( SKYFLATNAME )
// IF NO WOULD LEVELS, NOT EXIT
// terminates the sky texture to be used
// on the current episode, and the game version.
// code == commercial
// gamemode == pack (tr)
// gamemode == pack (plut)
// secrettext = false;
// secrettext = true;
// gamemode == go (completed);
// gamemode == go (nothing);

// skip the description field
// moment (check, 0, sizeof (check));
// sprint (check, "version %d", VERSION);
// if (strcmp (base, v, check))
// return;
// bad version
// save_p += VERSIONSIZE;

// This was quite messy with SPECIAL and commented parts.
// supposedly backs to make the latest edition work.
// it might not work properly.
// (episode < 1);
// episode = 1;

// if (gamemode == retail)
// {
//     episode = 1;
//     episode = 4;
// }
// else if (gamemode == shareware)
// {
//     if (episode > 3)
//     {
//         episode = 1; // only start episode 1 on shareware
//     }
//     else
//     {
//         if (episode > 3)
//         {
//             episode = 3;
//         }
//     }
// }

// if (map < 1)
//     map = 1;
// if (map > 9)
//     map = 9;
// M_ClearRandom (0);

// (skill == sk_nightmare || respawnparm)
// respawnmonsters = true;
// else
//     respawnmonsters = false;
// (lastparm || (skill == sk_nightmare && gameskill != sk_nightmare))
// for (i=S_SARG_RUN1; i<=S_SARG_PAIN2; i++)
//     mobinfo[MT_BRUISERSHOT].speed = 20*FRACUNIT;
// mobinfo[MT_TROOPSHOT].speed = 20*FRACUNIT;

// else if (skill != sk_nightmare && gameskill == sk_nightmare)
// for (i=S_SARG_RUN1; i<=S_SARG_PAIN2; i++)
//     mobinfo[MT_BRUISERSHOT].speed = 15*FRACUNIT;
// mobinfo[MT_TROOPSHOT].speed = 10*FRACUNIT;

// base_p += gameskill;
// base_p += gamemode;
// base_p += gamemode;
// for (i=0; i<MAXPLAYERS; i++)
//     {
//         if (i==0)
//         {
//             P_ArchivesPlayer (i);
//             P_ArchivesWard (i);
//             P_ArchivesBarkers (i);
//             P_ArchivesSpecials (i);
//         }
//         else
//         {
//             P_ArchivesPlayer (i);
//             P_ArchivesWard (i);
//             P_ArchivesBarkers (i);
//             P_ArchivesSpecials (i);
//         }
//     }

// if (secrettext)
//     wminfo.next = 8; // go to secret level
// else if (gamemode == 9)
// {
//     // returning from secret level
//     switch (gamemode)
//     {
//         case 1;
//             wminfo.next = 3;
//     }
}

```

Asynchronous and Parallel Programming in C#

Rasmus Lystrøm
External Associate Professor
ITU

Multithreading

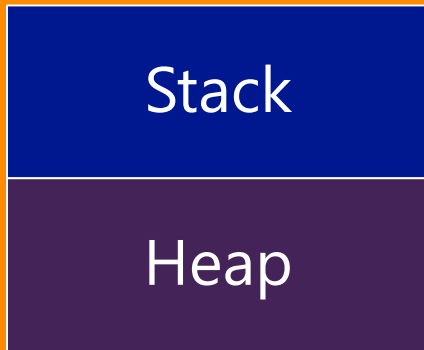
Enables executing several pieces of code simultaneously

- Leverage multicore CPUs
- Speed

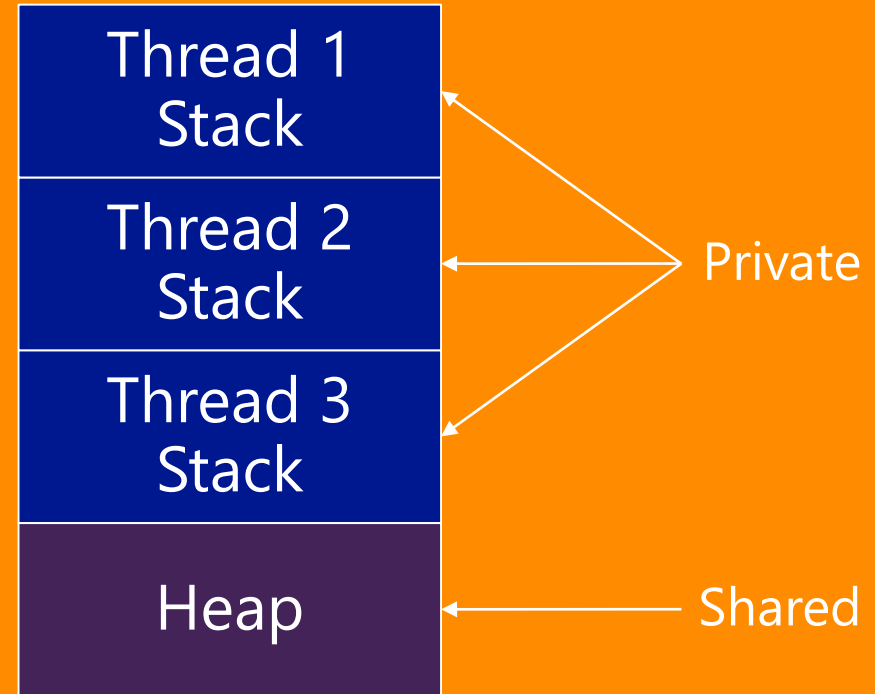
Concurrency

A property of systems in which several computations are executing **simultaneously**, and potentially interacting with each other. The computations may be executing on multiple cores in the same chip, preemptively time-shared threads on the same processor, or executed on physically separated processors.

Threads



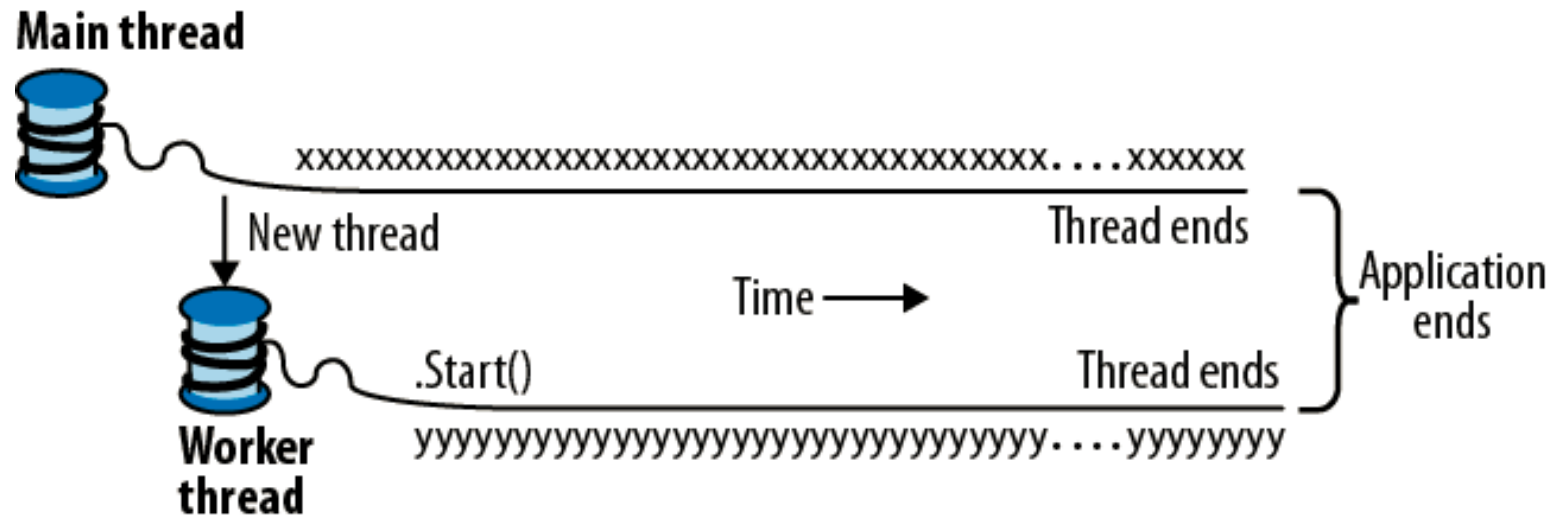
Single Threaded Program



Multithreaded Program

Threads Demo

Threads Example



© From C# 5.0 in a NUTSHELL

Race Condition



Race Condition

Behavior of a program where the output is dependent on the sequence or timing of other uncontrollable events.

→ Bug, when events do not happen in the order the programmer intended.

Race Condition Demo

Deadlock



Deadlock

A situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

Deadlock demo

Task Parallel Library

Task.Run

Task.Factory...

Task.Delay

Parallel.For

Parallel.ForEach

Parallel.Invoke

Parallel Linq → .AsParallel()

Task Parallel Library demo

System.Collections.Concurrent

ConcurrentQueue<T>

ConcurrentStack<T>

BlockingCollection<T>

ConcurrentDictionary<TKey, TValue>

Asynchronous Programming

async →

Method must return void, Task, or Task<T>

await →

Await method or task...

Note: Test methods must return Task

Async demo