Processes and Threads in C#

Understanding Concurrency Fundamentals

Mühlehner & Tavolato GmbH

What is a Process?

Processes

A process is an instance of a running program

Processes

A process is an instance of a running program

- **Independent memory space** - isolated from other processes

Processes

A process is an instance of a running program

- **Independent memory space** - isolated from other processes
- Has its own allocated system resources

Processes

A process is an instance of a running program

- **Independent memory space** - isolated from other processes
- Has its own allocated system resources
- Contains at least one thread (main thread)

Processes

A process is an instance of a running program

- **Independent memory space** - isolated from other processes
- Has its own allocated system resources
- Contains at least one thread (main thread)
- More resource-intensive to create and terminate

Processes

A process is an instance of a running program

- **Independent memory space** - isolated from other processes
- Has its own allocated system resources
- Contains at least one thread (main thread)
- More resource-intensive to create and terminate
- Examples: Each instance of Visual Studio, Chrome, or your C# application

Working with Processes in C#

```csharp
using System;
using System.Diagnostics;

// Start a new process
Process notepad = new Process();
notepad.StartInfo.FileName = "notepad.exe";
notepad.Start();

Console.WriteLine($"Started Notepad with PID: {notepad.Id}");

// Get current process info
Process current = Process.GetCurrentProcess();
Console.WriteLine($"Current process: {current.ProcessName}");
```

What is a Thread?

Threads

A thread is the smallest unit of execution within a process

Threads

A thread is the smallest unit of execution within a process

- **Share memory space** within their parent process

Threads

A thread is the smallest unit of execution within a process

- **Share memory space** within their parent process
- Lightweight compared to processes

Threads

A thread is the smallest unit of execution within a process

- **Share memory space** within their parent process
- Lightweight compared to processes
- Enable concurrent execution within a single process

Threads

A thread is the smallest unit of execution within a process

- **Share memory space** within their parent process
- Lightweight compared to processes
- Enable concurrent execution within a single process
- Share resources of their parent process

Threads

A thread is the smallest unit of execution within a process

- **Share memory space** within their parent process
- Lightweight compared to processes
- Enable concurrent execution within a single process
- Share resources of their parent process
- All threads in a process are terminated when the process ends

Thread Basics in C#

```
using System;
using System.Threading;

// Create a new thread
Thread worker = new Thread(() ⇒ {
    Console.WriteLine($"Worker thread ID: {Thread.CurrentThread.ManagedThreadId}");
    Console.WriteLine("Worker thread is running...");
    Thread.Sleep(1000);
    Console.WriteLine("Worker thread completed.");
});

// Start the thread
worker.Start();

// Main thread continues
Console.WriteLine($"Main thread ID: {Thread.CurrentThread.ManagedThreadId}");
```

Modern Threading with TPL

Task Parallel Library (TPL) - preferred approach in modern C#

```
using System;
using System.Threading.Tasks;

// Create and start a task
Task<int> task = Task.Run(() => {
    Console.WriteLine("Task is running calculation...");
    Task.Delay(1000).Wait();
    return 42;
});

// Do other work while task is running
Console.WriteLine("Main thread is doing other work...");

// Wait for result
int result = task.Result;
Console.WriteLine($"Task result: {result}");
```

Async/Await Pattern

The recommended way to handle asynchronous operations

```
using System.Threading.Tasks;

public async Task<string> FetchDataAsync()
{
    Console.WriteLine("Starting download...");

    // Simulate network request
    await Task.Delay(2000);

    Console.WriteLine("Download completed");
    return "Downloaded data";
}

// Usage
string data = await FetchDataAsync();
```

Process vs Thread

Process vs Thread

- Independent memory space

Process vs Thread

- Independent memory space
- Heavy resource footprint

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process
- Lightweight

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process
- Lightweight
- Easy data sharing

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process
- Lightweight
- Easy data sharing
- Limited isolation

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process
- Lightweight
- Easy data sharing
- Limited isolation
- Crash can affect entire process

Process vs Thread

- Independent memory space
- Heavy resource footprint
- Complex inter-process communication
- Strong isolation
- Crash doesn’t affect other processes
- Expensive context switching

- Shared memory within process
- Lightweight
- Easy data sharing
- Limited isolation
- Crash can affect entire process
- Faster context switching

When to Use What?

When to Use What?

Use Processes When:

When to Use What?

Use Processes When:

- Strong isolation is needed
- Running separate applications
- Memory protection is critical
- Resource sharing is not important

When to Use What?

Use Processes When:

- Strong isolation is needed
- Running separate applications
- Memory protection is critical
- Resource sharing is not important

Use Threads When:

When to Use What?

Use Processes When:

- Strong isolation is needed
- Running separate applications
- Memory protection is critical
- Resource sharing is not important

Use Threads When:

- Concurrent operations within same app
- Shared memory access required
- Performance and resource efficiency matter
- Implementing parallelism

THE END

Mühlehner & Tavolato GmbH