C# UI and Thread

Task – Wait – await – aync – Thread

陳偉凱

台北科大資工系

Motivation

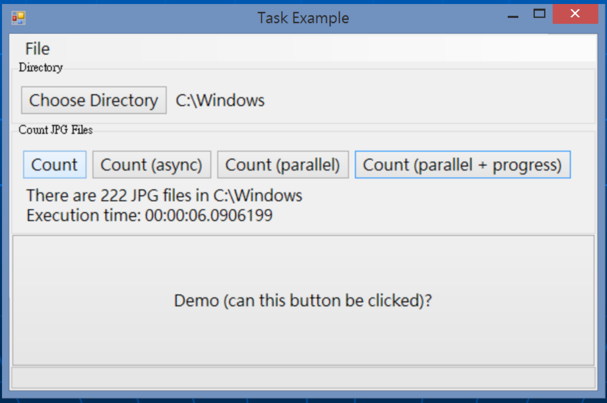
- Question 1: A program (method) P1 downloads a large file from the Internet. Is P1 I/O-bound or CPU bound?
- Question 2: A program (method) P2 determines whether a large integer is a prime number. Is P2 I/O-bound or CPU bound?

  Answer: P1 is I/O-bound. P2 is CPU bound.

- Question: Can you make P1 or P2 run faster?
  - No for P1
  - Yes for P2 (e.g., by computing divisions in parallel)

Motivation

- Question: What happens when you have an event handler that is extremely slow? For example, you would like to count the total number of JPG files under C:\Windows.



Task Example

File

Directory

Choose Directory C:\Windows

Count JPG Files

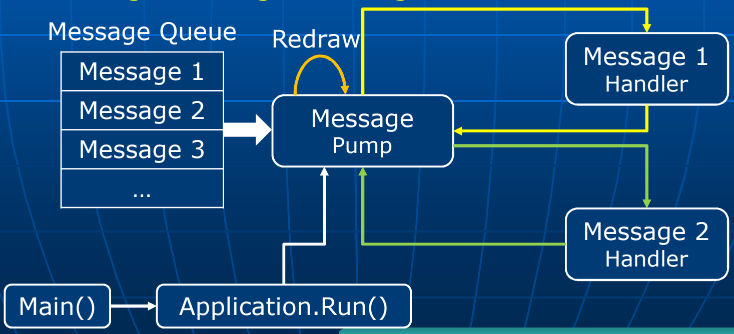
Count | Count (async) | Count (parallel) | Count (parallel + progress)

There are 222 JPG files in C:\Windows
Execution time: 00:00:06.0906199

Demo (can this button be clicked)?

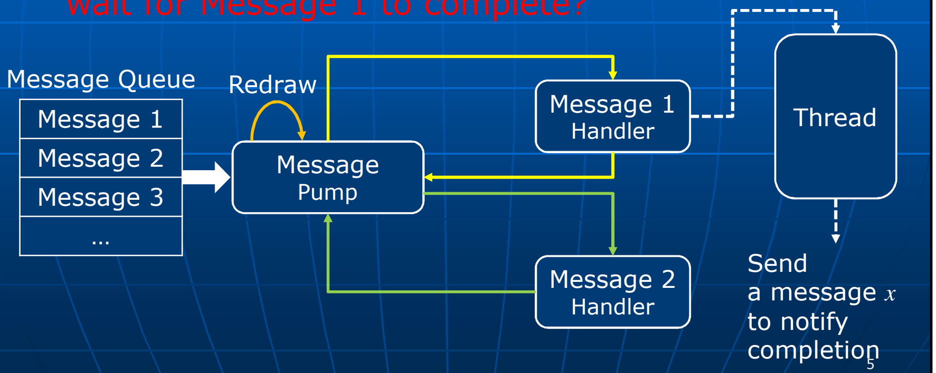
Message Pump

- Message Pump
  - Main() → Application.Run(…) → Message Pump
  - Message 2 (e.g., DemoButtonClicked) cannot be processed until message 1 is completed
  - A Message running for a long time blocks other messages



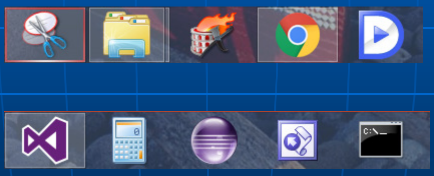
Message Pump

- Question: If Message 1 Handler needs a long running time, how to enable Message Pump to process Message 2 Handler without having to wait for Message 1 to complete?

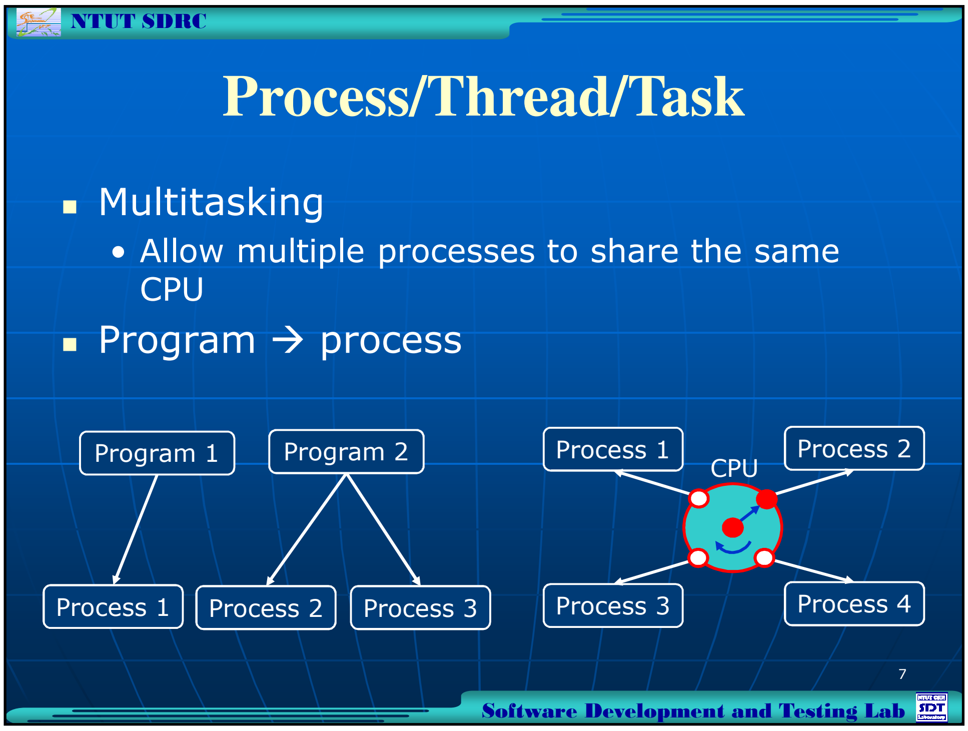


Process/Thread/Task

- Question: How does a CPU (single core) run several programs at the same time?



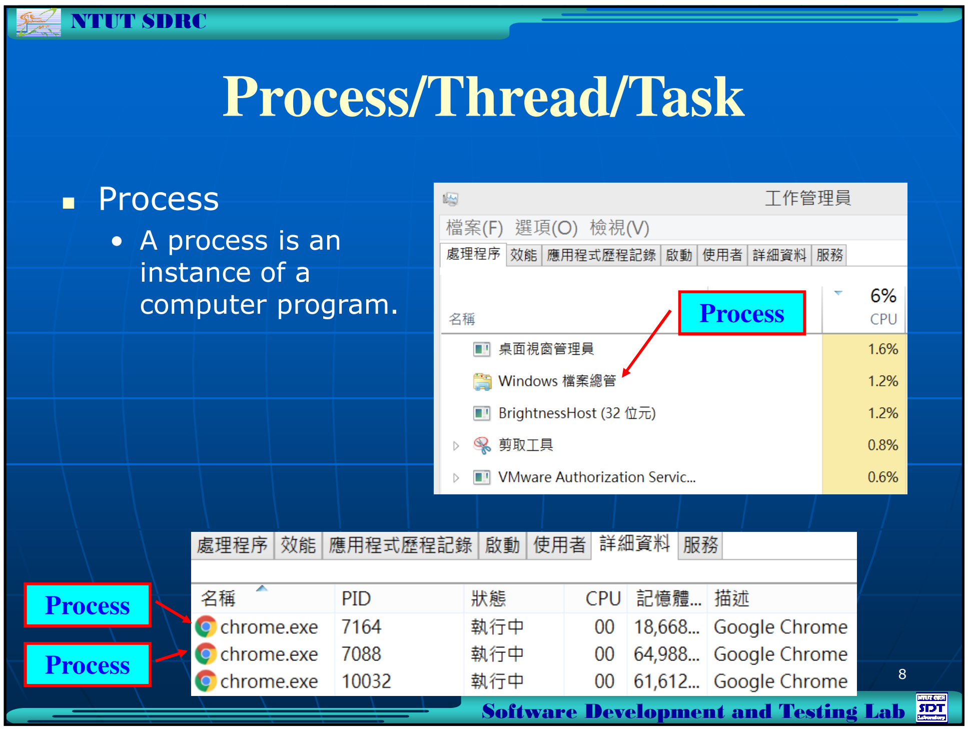
Multitasking: allow multiple processes to share the same CPU (next page)



Process/Thread/Task

- Multitasking
  - Allow multiple processes to share the same CPU
- Program → process

Program 1
Program 2
Process 1
Process 2
Process 3
Process 1
Process 2
CPU
Process 3
Process 4



Process/Thread/Task

- Process
  - A process is an instance of a computer program.

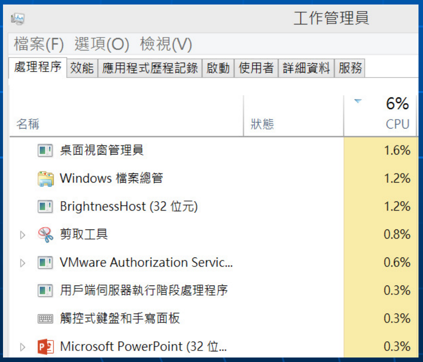
工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 啟動 使用者 詳細資料 服務
Process
Process
Process

名稱	6% CPU
桌面視窗管理員	1.6%
Windows 檔案總管	1.2%
BrightnessHost (32 位元)	1.2%
剪取工具	0.8%
VMware Authorization Servic...	0.6%

名稱	PID	狀態	CPU	記憶體...	描述
chrome.exe	7164	執行中	00	18,668...	Google Chrome
chrome.exe	7088	執行中	00	64,988...	Google Chrome
chrome.exe	10032	執行中	00	61,612...	Google Chrome

Process/Thread/Task

- Thread
  - A thread is contained inside a process. Multiple threads can exist within the same process and share resources (e.g., memory).

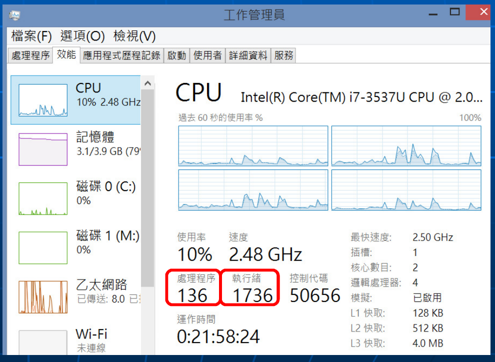


工作管理員

檔案(F) 選項(O) 檢視(V)

處理程序 | 效能 | 應用程式歷程記錄 | 啟動 | 使用者 | 詳細資料 | 服務

名稱	狀態	6% CPU
桌面視窗管理員		1.6%
Windows 檔案總管		1.2%
BrightnessHost (32 位元)		1.2%
剪取工具		0.8%
VMware Authorization Servic...		0.6%
用戶端伺服器執行階段處理程序		0.3%
觸控式鍵盤和手寫面板		0.3%
Microsoft PowerPoint (32 位...		0.3%



工作管理員

檔案(F) 選項(O) 檢視(V)

處理程序 | 效能 | 應用程式歷程記錄 | 啟動 | 使用者 | 詳細資料 | 服務

CPU 10% 2.48 GHz
記憶體 3.1/3.9 GB (79
磁碟 0 (C:) 0%
磁碟 1 (M:) 0%
乙太網路 已傳送: 8.0 已
Wi-Fi 未連線

CPU Intel(R) Core(TM) i7-3537U CPU @ 2.0...

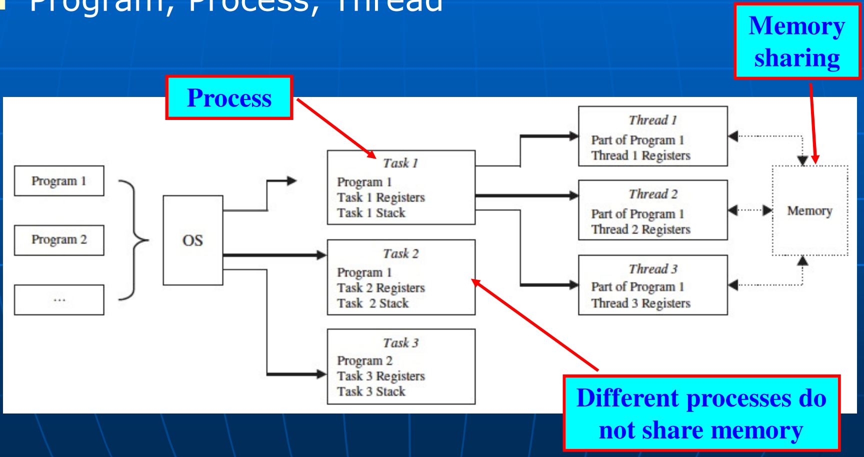
過去 60 秒的使用率 % 100%

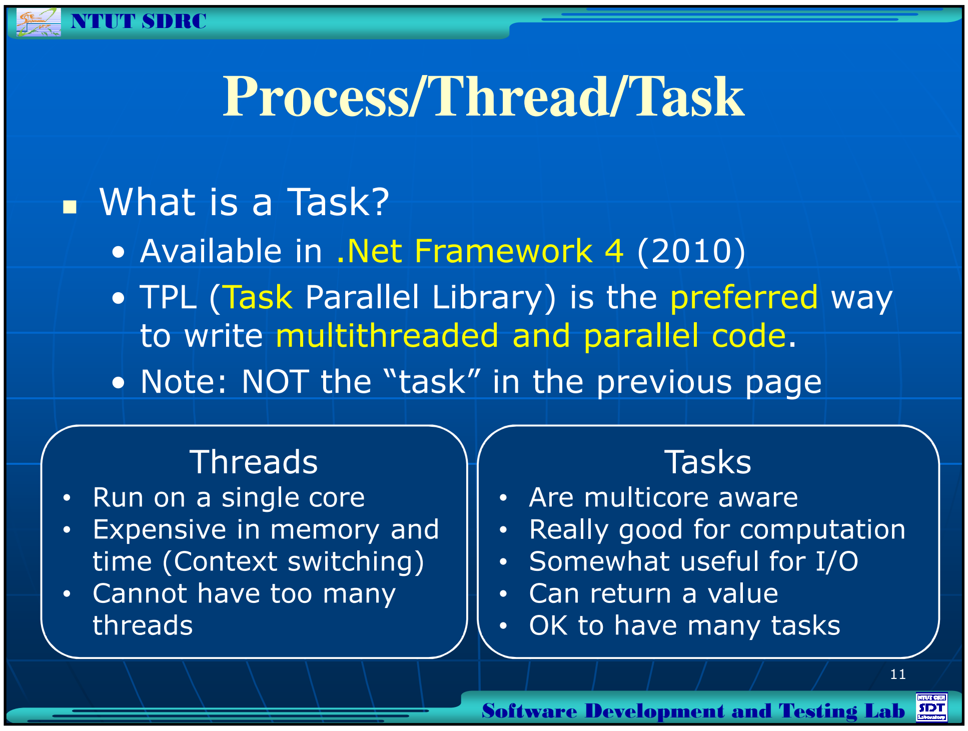
使用率 10%
速度 2.48 GHz
處理程序 136
執行緒 1736
控制代碼 50656
運作時間 0:21:58:24

最快速度:	2.50 GHz
插槽:	1
核心數目:	2
邏輯處理器:	4
模擬:	已啟用
L1 快取:	128 KB
L2 快取:	512 KB
L3 快取:	4.0 MB

Process/Thread/Task

- Program, Process, Thread



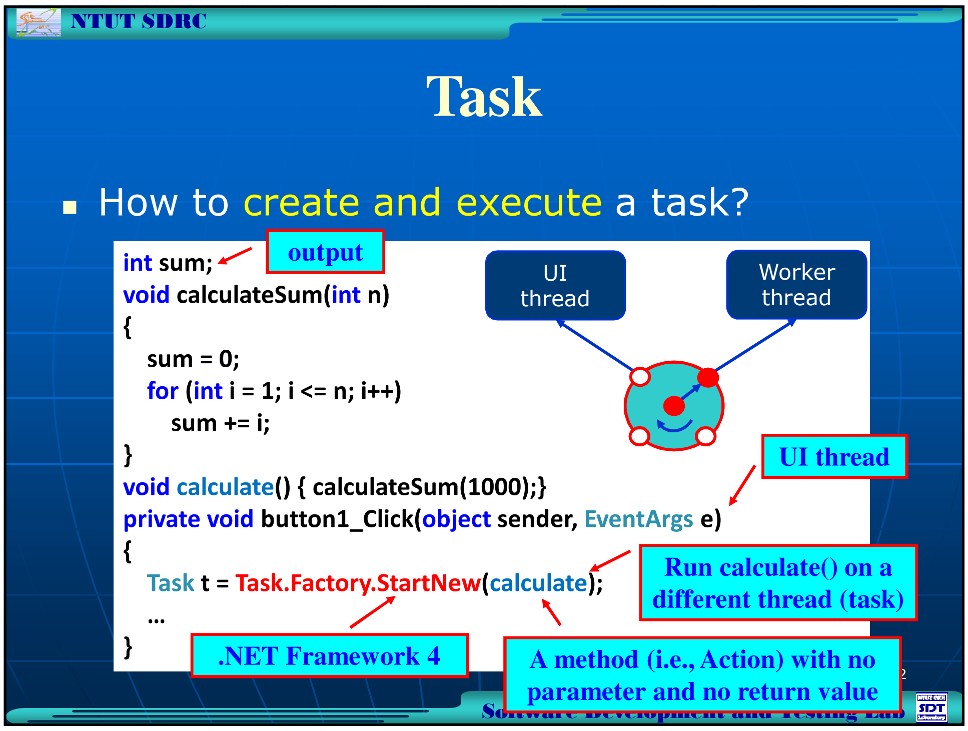


Process/Thread/Task

- What is a Task?
  - Available in .Net Framework 4 (2010)
  - TPL (Task Parallel Library) is the preferred way to write multithreaded and parallel code.
  - Note: NOT the "task" in the previous page

Threads
- Run on a single core
- Expensive in memory and time (Context switching)
- Cannot have too many threads

Tasks
- Are multicore aware
- Really good for computation
- Somewhat useful for I/O
- Can return a value
- OK to have many tasks

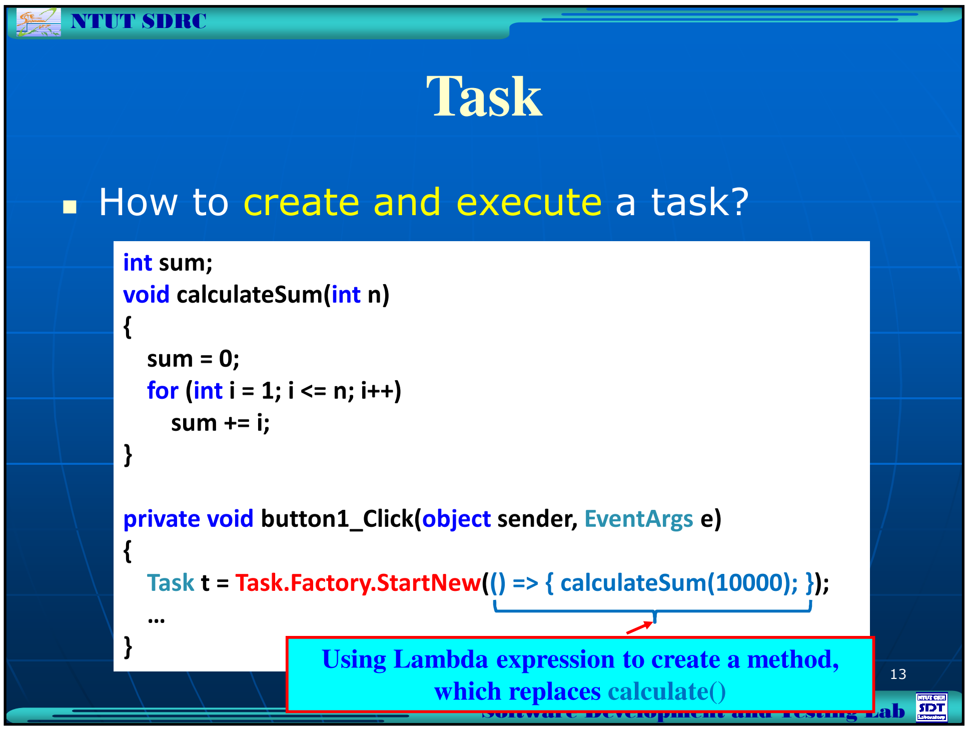


Task

- How to create and execute a task?

```
int sum;
void calculateSum(int n)
{
   sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;
}
void calculate() { calculateSum(1000);}
private void button1_Click(object sender, EventArgs e)
{
   Task t = Task.Factory.StartNew(calculate);
   …
}
```

output
UI thread
Worker thread
UI thread
Run calculate() on a different thread (task)
.NET Framework 4
A method (i.e., Action) with no parameter and no return value



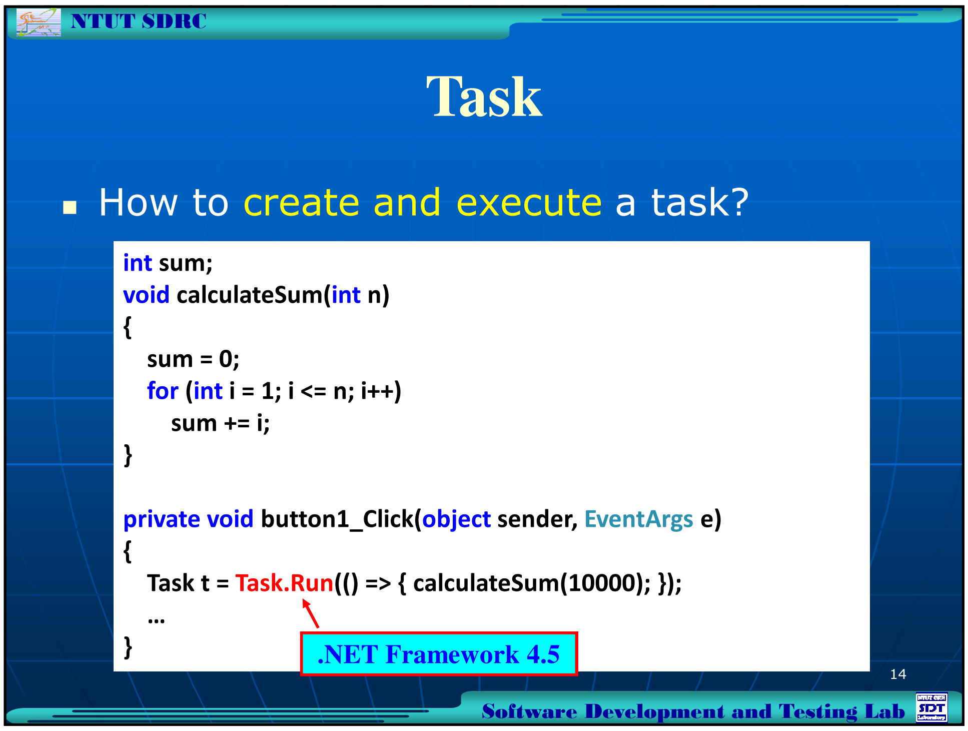
Task

- How to create and execute a task?

```
int sum;
void calculateSum(int n)
{
   sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;
}

private void button1_Click(object sender, EventArgs e)
{
   Task t = Task.Factory.StartNew(() => { calculateSum(10000); });
   …
}
```

Using Lambda expression to create a method, which replaces calculate()



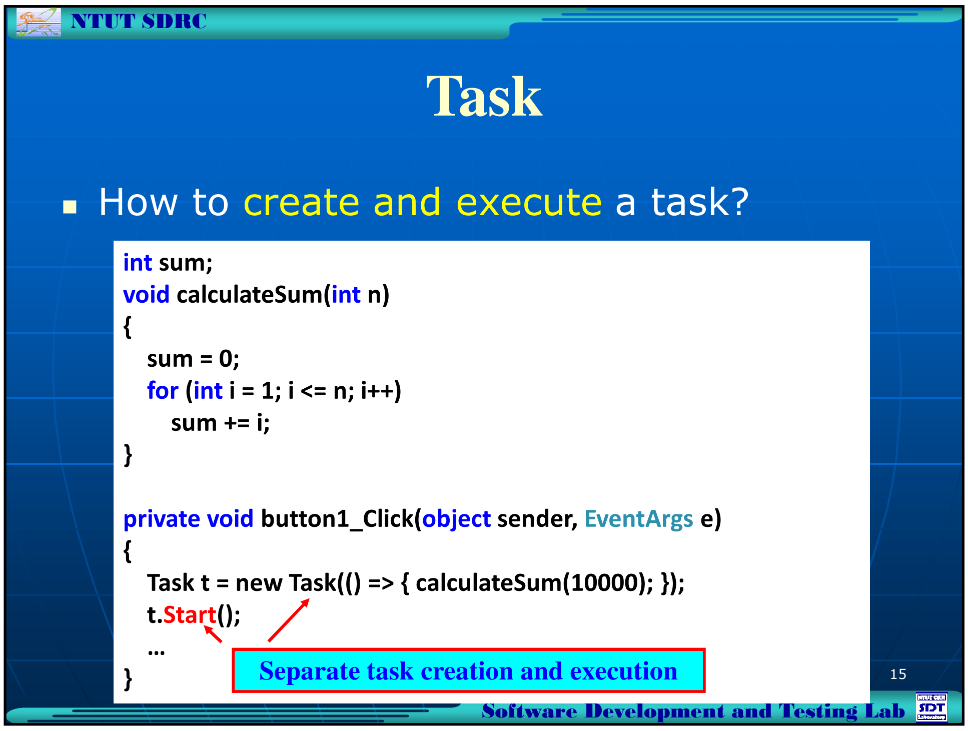
Task

- How to create and execute a task?

```
int sum;
void calculateSum(int n)
{
   sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;
}

private void button1_Click(object sender, EventArgs e)
{
   Task t = Task.Run(() => { calculateSum(10000); });
   …
}
```

.NET Framework 4.5



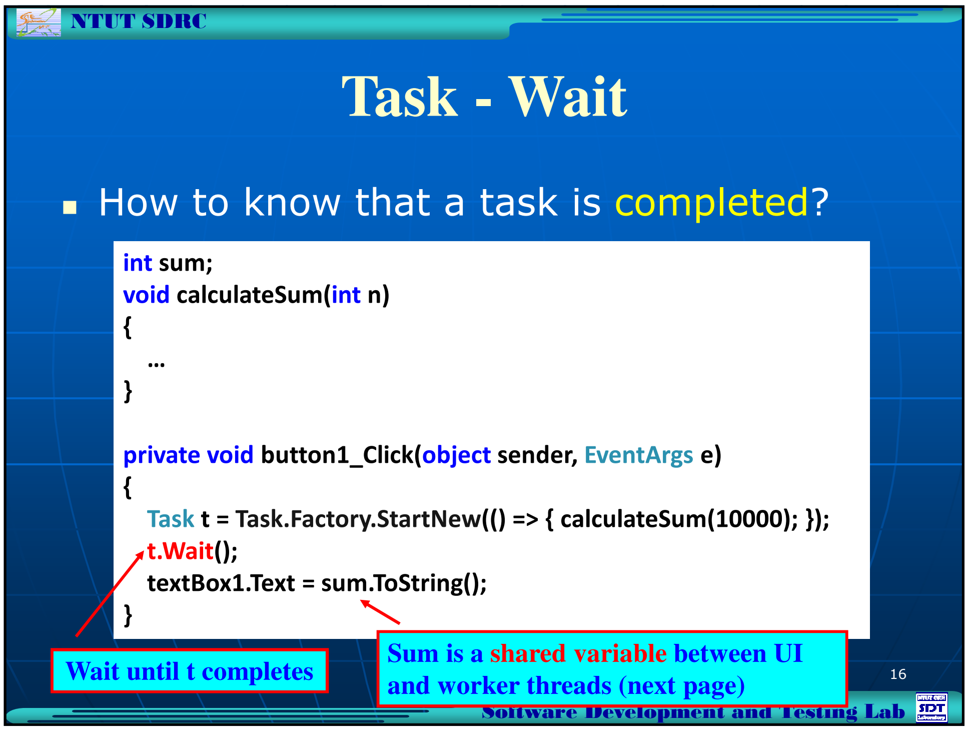
Task

- How to create and execute a task?

```
int sum;
void calculateSum(int n)
{
    sum = 0;
    for (int i = 1; i <= n; i++)
        sum += i;
}

private void button1_Click(object sender, EventArgs e)
{
    Task t = new Task(() => { calculateSum(10000); });
    t.Start();
    …
}
```

Separate task creation and execution



Task - Wait

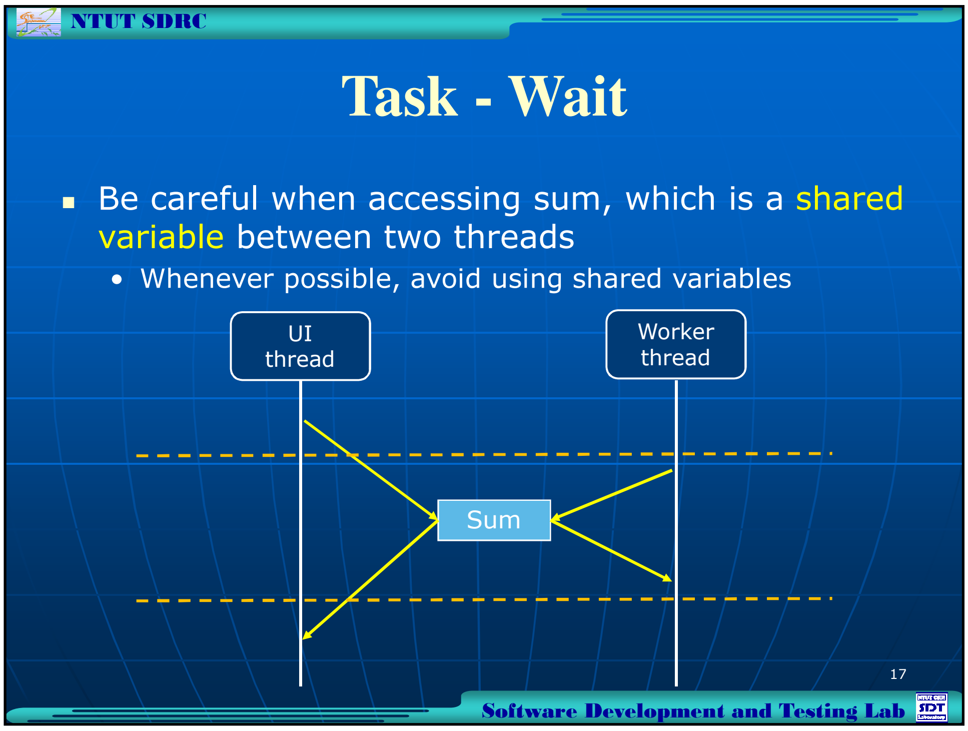
- How to know that a task is completed?

```
int sum;
void calculateSum(int n)
{
    …
}

private void button1_Click(object sender, EventArgs e)
{
    Task t = Task.Factory.StartNew(() => { calculateSum(10000); });
    t.Wait();
    textBox1.Text = sum.ToString();
}
```

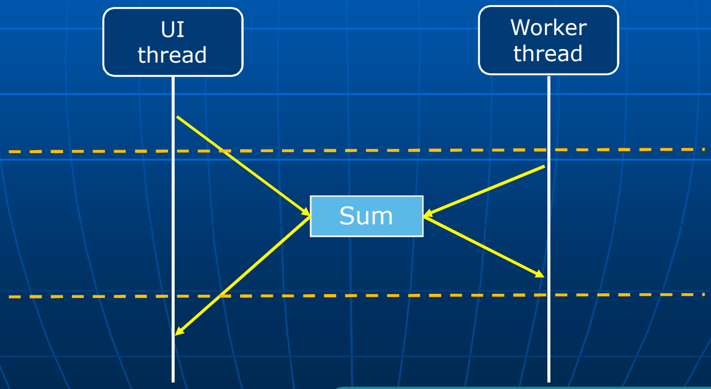
Wait until t completes

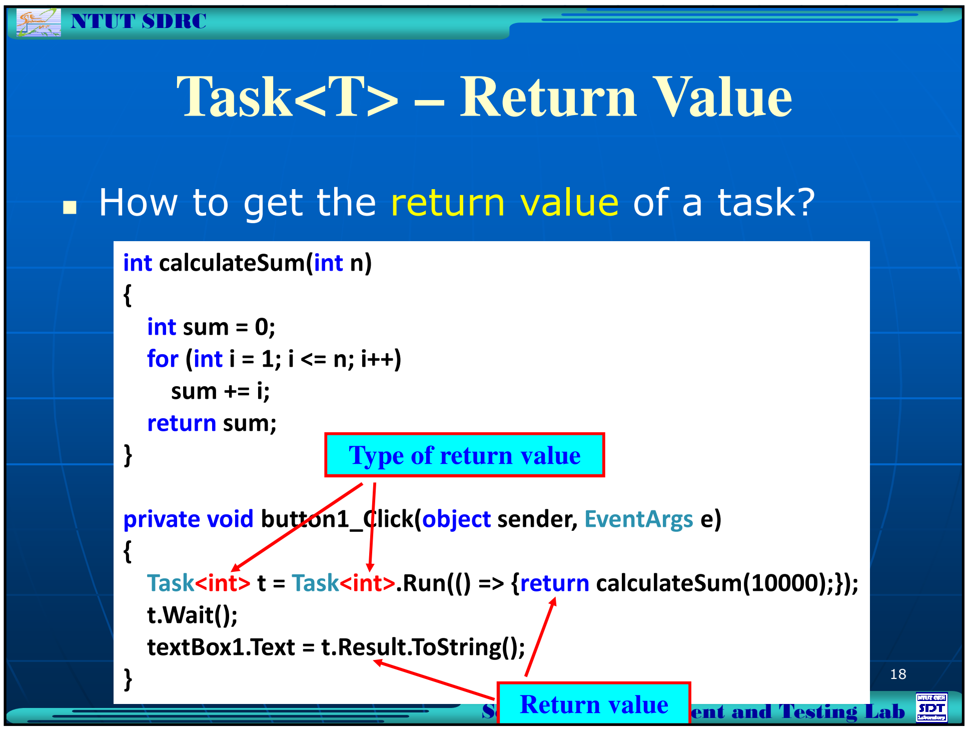
Sum is a shared variable between UI and worker threads (next page)



Task - Wait

- Be careful when accessing sum, which is a shared variable between two threads
  - Whenever possible, avoid using shared variables





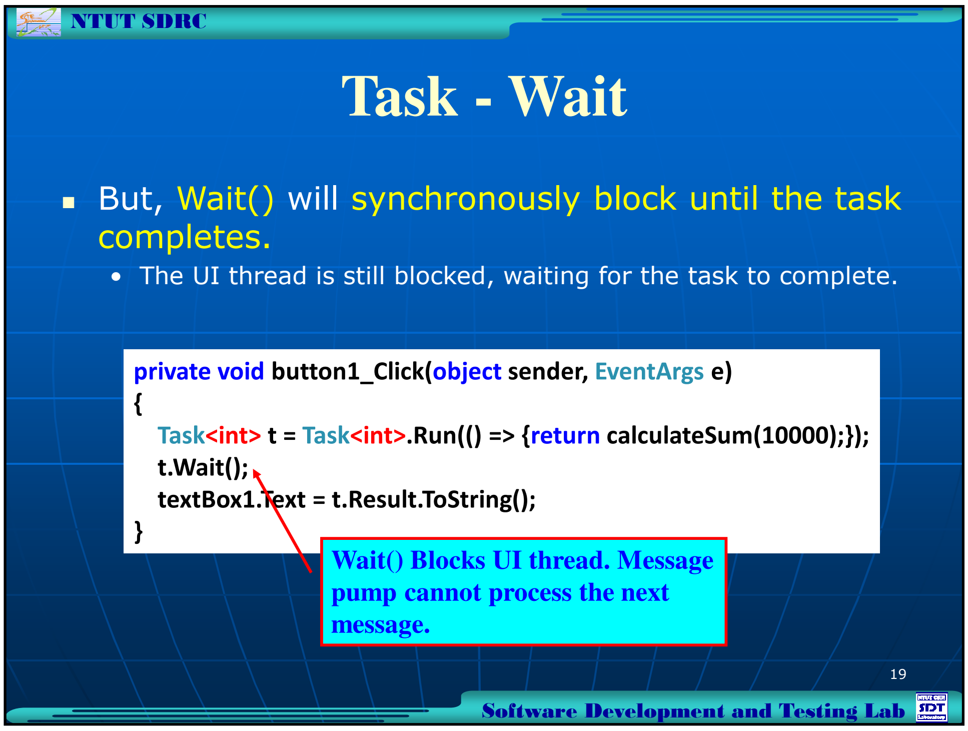
Task<T> – Return Value

- How to get the return value of a task?

```
int calculateSum(int n)
{
   int sum = 0;
   for (int i = 1; i <= n; i++)
      sum += i;
   return sum;
}

private void button1_Click(object sender, EventArgs e)
{
   Task<int> t = Task<int>.Run(() => {return calculateSum(10000);});
   t.Wait();
   textBox1.Text = t.Result.ToString();
}
```

Type of return value
Return value

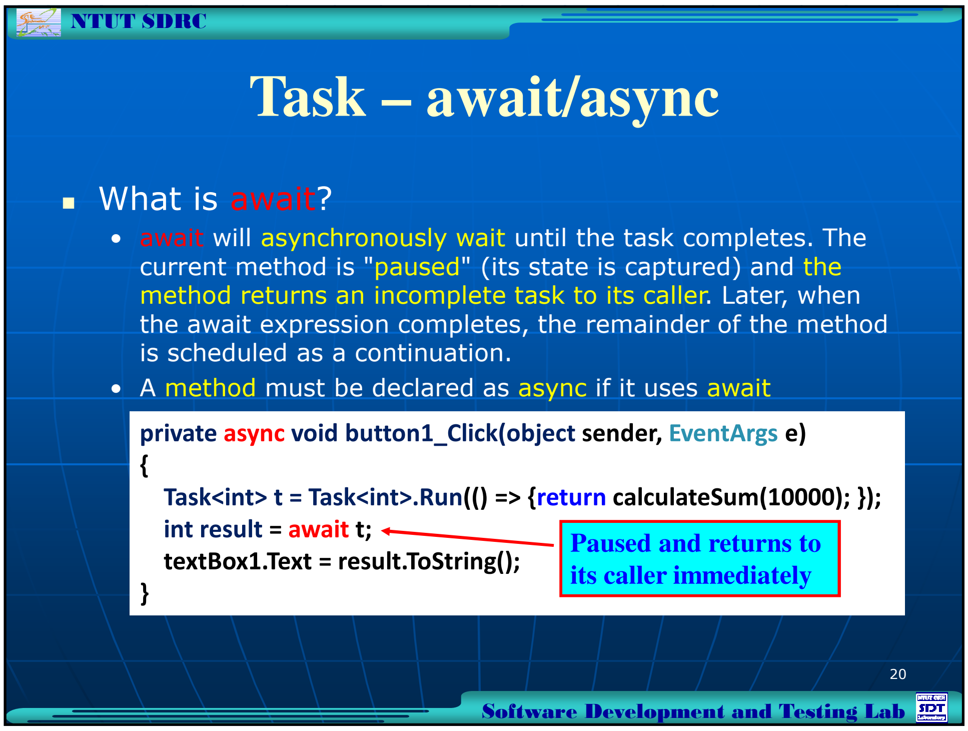


Task - Wait

- But, Wait() will synchronously block until the task completes.
  - The UI thread is still blocked, waiting for the task to complete.

```
private void button1_Click(object sender, EventArgs e)
{
   Task<int> t = Task<int>.Run(() => {return calculateSum(10000);});
   t.Wait();
   textBox1.Text = t.Result.ToString();
}
```

Wait() Blocks UI thread. Message pump cannot process the next message.



Task – await/async

- What is await?
  - await will asynchronously wait until the task completes. The current method is "paused" (its state is captured) and the method returns an incomplete task to its caller. Later, when the await expression completes, the remainder of the method is scheduled as a continuation.
  - A method must be declared as async if it uses await

```
private async void button1_Click(object sender, EventArgs e)
{
   Task<int> t = Task<int>.Run(() => {return calculateSum(10000); });
   int result = await t;
   textBox1.Text = result.ToString();
}
```

Paused and returns to its caller immediately

Using an async method

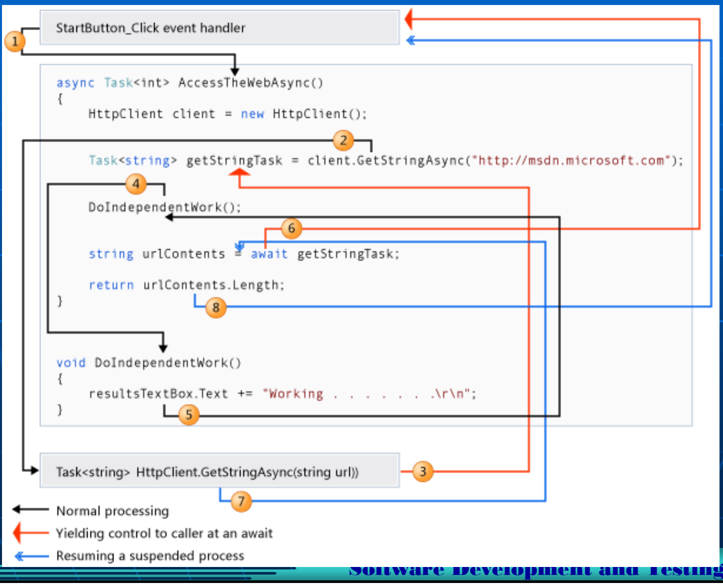
- How to call an async method?

async

```
private async void ClickDownloadButton(object sender, RoutedEventArgs e)
{
        _textBox.Text = "";
        string uri = "http://www.ntut.edu.tw/~wkchen/TPL_ICS2002.pdf";
        var client = new HttpClient();
        HttpResponseMessage response = await client.GetAsync(uri);
        if (response.StatusCode == HttpStatusCode.OK)
           _textBox.Text = await response.Content.ReadAsStringAsync();
        else
           throw new Exception("Error: " + response.StatusCode);
}
```

Many C# libraries offer async methods

Control Flow



StartButton_Click event handler

```
async Task<int> AccessTheWebAsync()
{
    HttpClient client = new HttpClient();

    Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");

    DoIndependentWork();

    string urlContents = await getStringTask;

    return urlContents.Length;
}

void DoIndependentWork()
{
    resultsTextBox.Text += "Working . . . . . . .\r\n";
}
```

Task<string> HttpClient.GetStringAsync(string url))

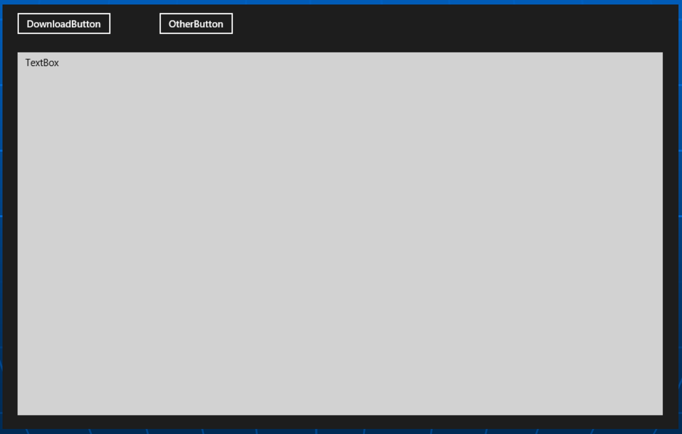
← Normal processing

← Yielding control to caller at an await

← Resuming a suspended process

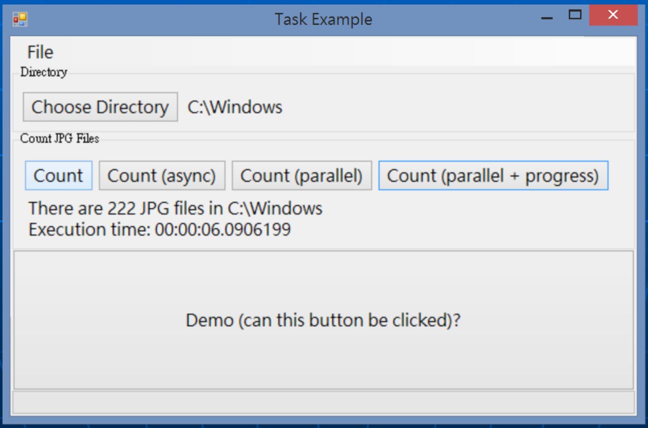
Example

- Example: HttpDownloadAsyncApp



Example

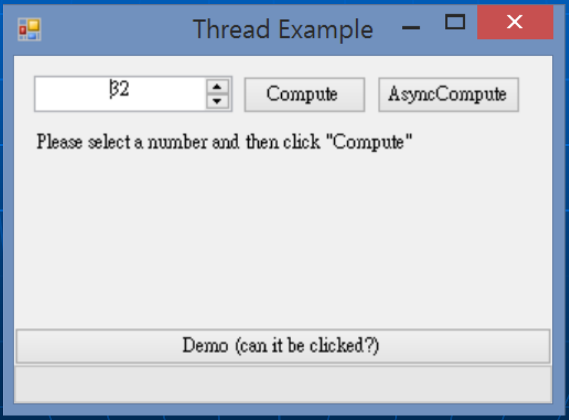
- Example: Task Example



There are 222 JPG files in C:\Windows
Execution time: 00:00:06.0906199

Thread Example

- Example: using traditional threads



Please select a number and then click "Compute"