What is the difference between concurrency, parallelism and asynchronous methods?

Asked 9 years, 10 months ago Active 2 months ago Viewed 66k times

205

Concurrency is having two tasks run in parallel on separate threads. However, asynchronous methods run in parallel but on the same 1 thread. How is this achieved? Also, what about parallelism?

What are the differences between these 3 concepts?

119

multithreading asynchronous concurrency

edited May 12 '16 at 17:03

Laurel
5,402 10 23 46

asked Jan 30 '11 at 18:22

GurdeepS
57.4k 94 231 366

9 The term "asynchronous" can mean a lot of different things. Those terms are related, but they do not describe disjoint sets of things. The meanings overlap and vary by situation. – Pointy Jan 30 '11 at 18:26

2 So first concurrency is running two or more processes at the same time. With that out of the way, being concurrent is not being parallel. Parallel processes require two or more cores whereas concurrent processes can time share a single core. – Rick O'Shea Jul 28 '18 at 1:33

13 Answers

Active | Oldest | Votes

148

Concurrent and parallel are effectively the same principle as you correctly surmise, both are related to tasks being executed simultaneously although I would say that parallel tasks should be truly multitasking, executed "at the same time" whereas concurrent could mean that the tasks are sharing the execution thread while still appearing to be executing in parallel.

Asynchronous methods aren't directly related to the previous two concepts, asynchrony is used to present the impression of concurrent or parallel tasking but effectively an asynchronous method call is normally used for a process that needs to do work away from the current application and we don't want to wait and block our application awaiting the response.

For example, getting data from a database could take time but we don't want to block our UI waiting for the data. The async call takes a call-back reference and returns execution back to your code as soon as the request has been placed with the remote system. Your UI can continue to respond to the user while the remote system does whatever processing is required, once it returns the data to your call-back method then that method can update the UI (or handoff that update) as appropriate.

From the User perspective, it appears like multitasking but it may not be.

EDIT

It's probably worth adding that in many implementations an asynchronous method call will cause a thread to be spun up but it's not essential, it really depends on the operation being executed and how the response can be notified back to the system.

edited Jan 21 '19 at 7:01

Ijas Ameenudeen
8,055 3 34 49

answered Jan 30 '11 at 18:48

Lazarus
37.1k 4 39 53

41 I'd argue that you have parallelism and concurrency mixed up in your first paragraph. Concurrency refers managing multiple threads of execution, where parallelism is more specifically, multiple threads of execution executing simultaneously. Concurrency is the broader term which can encompass parallelism. – Mark H Jan 30 '11 at 19:37

7 While the two words are very similar and could be confused (and often are), they do have different definitions: Concurrent = Existing, happening, or done at the same time. Parallel = of or pertaining to the apparent or actual performance of more than one operation at a time, by the same or different devices. As you can see parallel does not necessarily mean concurrent but could be just appearing to be concurrent. At the end of the day, the words are often used interchangeably and with *n* dev is a room you'll probably get *n+1* definitions ;) – Lazarus Jan 31 '11 at 15:12

3 @Mehrdad If you are going to base your definitions of dictionary words on their use in marketing materials then I think you are likely to find yourself at somewhat of a disadvantage. – Lazarus Nov 27 '12 at 8:27

9 Wrong. In the context of programming, concurrency is the ability of your code to be "composed" into bits of logic that *could* be run at the same time. Parallelism (when combined with concurrency) is taking said code and running it on a 100-core machine. – Frank Radocaj Feb 23 '14 at 1:11

3 @FrankRadocaj has it right. Concurrent means that the program can be split up into units (units essentially being threads) that can be ran in any order and have a determinate outcome. Parallel means these units/threads are being ran literally at the same time on multiple processors. – user7917402 Apr 26 '17 at 9:23

98

In Short,

Concurrency means multiple tasks which start, run, and complete in overlapping time periods, in no specific order. Parallelism is when multiple tasks OR several part of a unique task literally run at the same time, e.g. on a multi-core processor.

Remember that Concurrency and parallelism are NOT the same thing.

Differences between concurrency vs. parallelism

Now let's list down remarkable differences between concurrency and parallelism.

Concurrency is when two tasks can start, run, and complete in overlapping time periods. Parallelism is when tasks literally run at the same time, eg. on a multi-core processor.

Concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations.

Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

An application can be concurrent – but not parallel, which means that it processes more than one task at the same time, but no two tasks are executing at same time instant.

An application can be parallel – but not concurrent, which means that it processes multiple sub-tasks of a task in multi-core CPU at same time.

An application can be neither parallel – nor concurrent, which means that it processes all tasks one at a time, sequentially.

An application can be both parallel – and concurrent, which means that it processes multiple tasks concurrently in multi-core CPU at same time.

Concurrency

> Concurrency is essentially applicable when we talk about minimum two tasks or more. When an application is capable of executing two tasks virtually at same time, we call it concurrent application. Though here tasks run looks like simultaneously, but essentially they MAY not. They take advantage of CPU time-slicing feature of operating system where each task run part of its task and then go to waiting state. When first task is in waiting state, CPU is assigned to second task to complete it's part of task.
>
> Operating system based on priority of tasks, thus, assigns CPU and other computing resources e.g. memory; turn by turn to all tasks and give them chance to complete. To end user, it seems that all tasks are running in parallel. This is called concurrency.

Parallelism

> Parallelism does not require two tasks to exist. It literally physically run parts of tasks OR multiple tasks, at the same time using multi-core infrastructure of CPU, by assigning one core to each task or sub-task.
>
> Parallelism requires hardware with multiple processing units, essentially. In single core CPU, you may get concurrency but NOT parallelism.

Asynchronous methods

This is not related to Concurrency and parallelism, asynchrony is used to present the impression of concurrent or parallel tasking but effectively an asynchronous method call is normally used for a process that needs to do work away from the current application and we don't want to wait and block our application awaiting the response.

edited Apr 13 '16 at 16:47

answered Apr 13 '16 at 16:35

Vipin Jain
4,828 18 35

2 Very helpful. Thanks – Neha Feb 8 '18 at 18:17

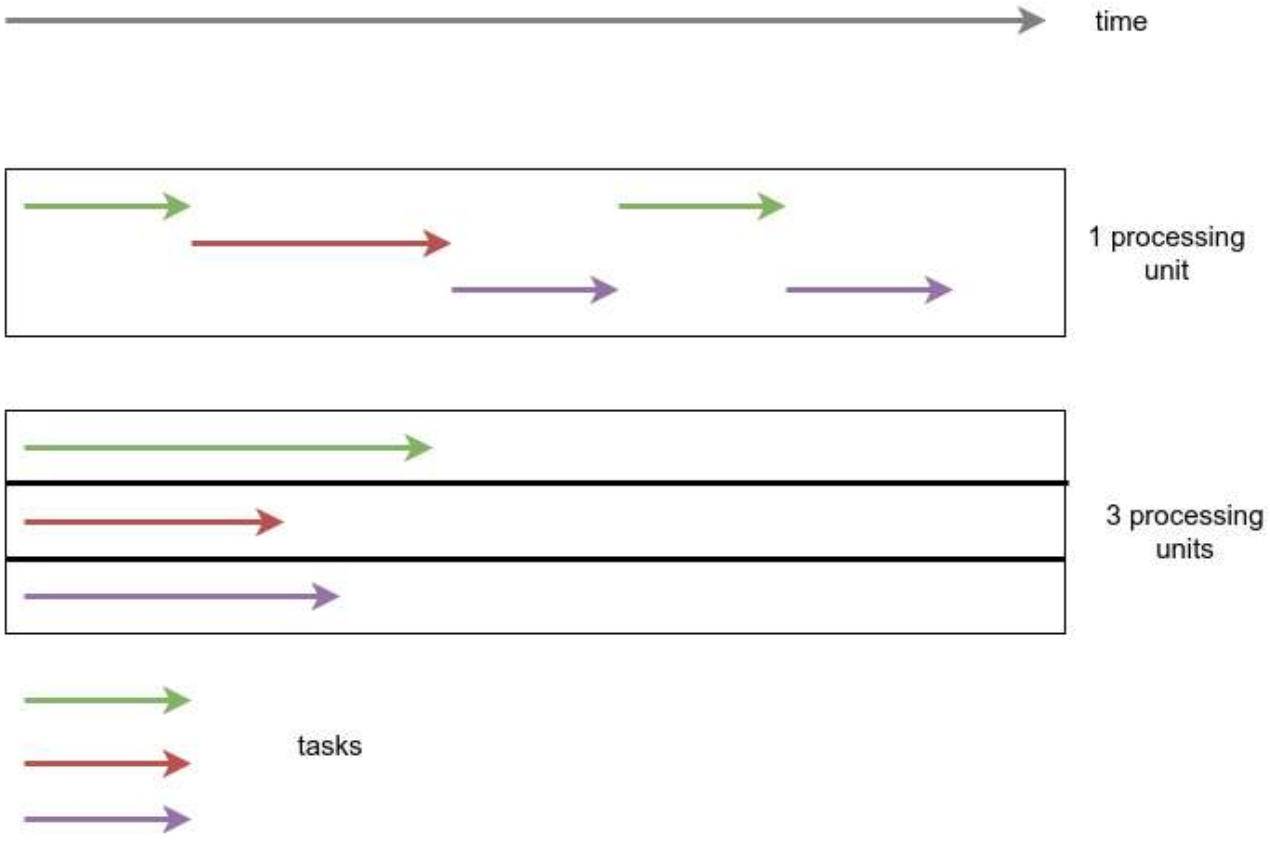
I am still confused about Async vs concurrency. I have seen the phrase "running tasks asynchronously" and "running tasks concurrently" used. Running tasks asynchronous and running tasks concurrently seem to mean the same thing? – Moondra Mar 24 '19 at 19:31

1 @Moondra Async task runs on a single thread, nothing related to multithreading. async task sends a task to an external process outside your application... i.e database, file reader... these are IO processes then a callback is added on the process to perform an action when the IO process is finished. what i'm not sure of is that there are some "async" features in some programming language that creates a separate new thread when you call an async task, i heard C# is one but i'm not sure about thus – Tobi Owolawi Sep 20 '19 at 14:21

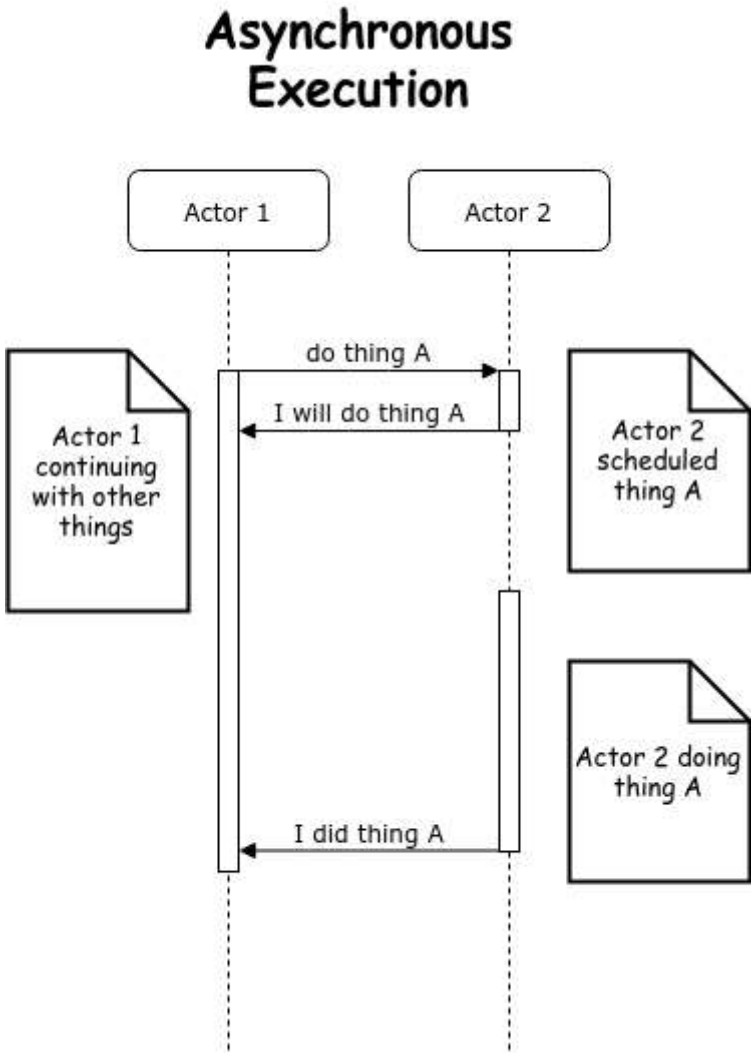
71

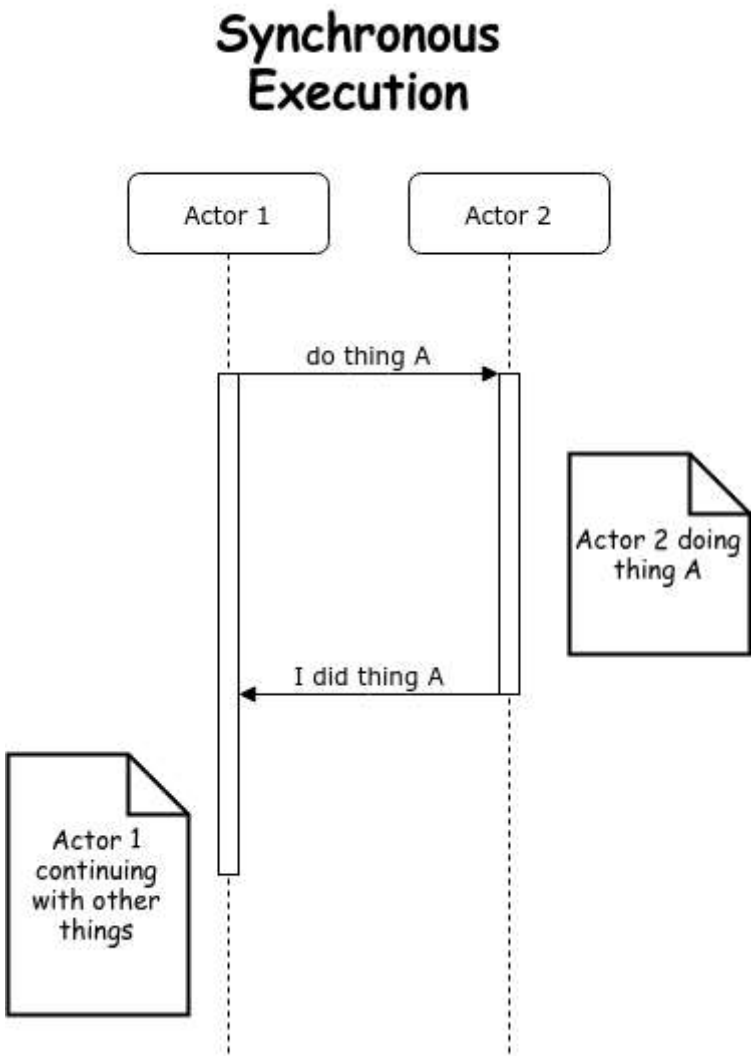
Concurrency is when the execution of multiple tasks is interleaved, instead of each task being executed sequentially one after another.

Parallelism is when these tasks are actually being executed in parallel.



Asynchrony is a separate concept (even though related in some contexts). It refers to the fact that one event might be happening at a different time (not in synchrony) to another event. The below diagrams illustrate what's the difference between a synchronous and an asynchronous execution, where the actors can correspond to different threads, processes or even servers.





edited Feb 18 '19 at 19:03

answered Jan 30 '18 at 20:50

Dimos
6,628 31 32

8 Simple, effective illustration. – contactmatt Mar 12 '18 at 15:33

1 is concurrency the same as asynchrony? – nos Feb 18 '19 at 3:52

2 These 2 concepts are very close, indeed, but not the same. In practice, asynchrony is more related with the interaction between actions (say A & B), where one (B) is triggered by the other (A) and whether the second one will wait on the first one to complete. Concurrency is a more general terms for actions that can also be unrelated to each other and whether they are executed in sequence or their executions are interleaved. – Dimos Feb 18 '19 at 19:22

1 So asynchrony is mostly about blocking and non blocking – Daniel May 22 at 17:46

20

There are several scenarios in which concurrency can occur:

Asynchrony— This means that your program performs non-blocking operations. For example, it can initiate a request for a remote resource via HTTP and then go on to do some other task while

it waits for the response to be received. It's a bit like when you send an email and then go on with your life without waiting for a response.

Parallelism— This means that your program leverages the hardware of multi-core machines to execute tasks at the same time by breaking up work into tasks, each of which is executed on a separate core. It's a bit like singing in the shower: you're actually doing two things at exactly the same time.

Multithreading— This is a software implementation allowing different threads to be executed concurrently. A multithreaded program appears to be doing several things at the same time even when it's running on a single-core machine. This is a bit like chatting with different people through various IM windows; although you're actually switching back and forth, the net result is that you're having multiple conversations at the same time.

answered Dec 18 '18 at 18:14

rahulaga_dev
2,730 3 20 36

These are really good analogies! Thanks. Would it be fair to say concurrency can be defined with your multithreading definition? So concurrency = multithreading on a single-core which looks like it's happening at the same time but it's really switching back and forth really quickly? – wongz Jun 7 at 14:46

1 Simple and to the point – Guzman Ojero Dec 3 at 18:52

18

Everyone is having trouble associating asynchronous to either parallelism or concurrency because asynchronous is not an antonym to either parallel or concurrent. It is an antonym of Synchronous. Which just indicates if something, in this case threads, will be synched with something else, in this case another thread.

answered Mar 14 '14 at 19:45

Aloysius Snuffleupagus
181 1 2

8

Concurrency means executing multiple tasks at the same time, but not necessarily simultaneously. When you have to perform more than one task but you have a single resource then we go for concurrency. In a single core environment, concurrency is achieved by context switching.

Parallelism is like performing more than one task simultaneously, like you can sing and bathe together. Now you are doing the tasks in parallel.

The term *asynchronous* is related to thread execution. In an asynchronous model, when one task gets executed, you can switch to a different task without waiting for the previous task to get completed.

Asynchronous programming helps us to achieve concurrency. Asynchronous programming in a multi-threaded environment is a way to achieve parallelism.

edited Oct 12 at 16:05

Rob Bednark
18.7k 18 66 95

answered Apr 15 at 9:30

Dhirendra Gautam
393 3 9

5

"Sync and async are programming models. Concurrent and parallel are ways tasks are executed...". Source: https://medium.com/better-programming/sync-vs-async-vs-concurrent-vs-parallel-5754cdb60f66

In other words, sync and async describe how your program executes when making a function call (will it wait or will it continue executing?), whilst concurrent and parallel describe how a function (a task) will be executed (concurrent = possibly executed at the same time, parallel = effectively executed at the same time).

edited Jun 17 at 10:06

answered Dec 17 '19 at 8:40

Pedro Boechat
1,947 17 21

medium is not a source, it is an article by someone else talking about his (mis)understanding of a subject, it doesn't make him/her an authority. – Moha the almighty camel Feb 29 at 22:12

3 It's a citation, therefore the source. Most answers here are not written by authorities in any field and the explanation the author gave is good enough. – Pedro Boechat Feb 29 at 23:03

At stackoverflow you at least have a voting system, and it's a community of professionals. Anyone can write anything on medium. It's not a fair comparison between the two. – Moha the almighty camel Mar 1 at 7:06

3 Anyone can write anything here too, I don't know why you're picking on Medium. Anyway, I'm a professional programmer and I endorse this understanding. And I find it elegant because its relatively short. – Pedro Boechat Mar 1 at 8:50

The problem is not that Medium is a bad source, the problem is that an answer that merely links to an external page is not an answer. You could greatly improve this answer if you elaborated on the quoted portion or summarized the contents of the article rather than just linking to it. – David Schwartz Jun 16 at 19:47

4

Concurrency

Concurrency means that an application is making progress on more than one task at the same time (concurrently). Well, if the computer only has one CPU the application may not make progress on more than one task at exactly the same time, but more than one task is being processed at a time inside the application. It does not completely finish one task before it begins the next.

Parallelism

Parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time.

Concurrency vs. Parallelism In Detail

As you can see, concurrency is related to how an application handles multiple tasks it works on. An application may process one task at at time (sequentially) or work on multiple tasks at the same time (concurrently).

Parallelism on the other hand, is related to how an application handles each individual task. An application may process the task serially from start to end, or split the task up into subtasks which can be completed in parallel.

As you can see, an application can be concurrent, but not parallel. This means that it processes more than one task at the same time, but the tasks are not broken down into subtasks.

An application can also be parallel but not concurrent. This means that the application only works on one task at a time, and this task is broken down into subtasks which can be processed in parallel.

Additionally, an application can be neither concurrent nor parallel. This means that it works on only one task at a time, and the task is never broken down into subtasks for parallel execution.

Finally, an application can also be both concurrent and parallel, in that it both works on multiple tasks at the same time, and also breaks each task down into subtasks for parallel execution. However, some of the benefits of concurrency and parallelism may be lost in this scenario, as the CPUs in the computer are already kept reasonably busy with either concurrency or parallelism alone. Combining it may lead to only a small performance gain or even performance loss. Make sure you analyze and measure before you adopt a concurrent parallel model blindly.

From http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html

edited Jun 20 at 9:12

Community ♦
1 1

answered Aug 6 '17 at 4:42

LONGHORN007
476 7 22

4

There's a bit of semantics to clear up here:

Concurrency or Parallelism is a question of **resource contention**, whereas Asynchronous is about **control flow**.

Different **procedures** (or their constituent **operations**) are termed Asynchronous, when there's no deterministic implementation of the the order of their **processing**; in other words, there's a probability that any of them could be processed at any given time T. By definition, multiple processors (e.g. CPUs or Persons) make it possible for several of them to be processed at the same time; on a single processor, their processing is interleaved (e.g. Threads).

Asynchronous procedures or operations are termed Concurrent, when they **share resources**; Concurrency is the definite possibility of contention at any given time T. Parallelism is trivially guaranteed when no resources are shared (e.g. different processor and storage); otherwise Concurrency control must be addressed.

Hence an Asynchronous procedure or operation may be processed in Parallel or Concurrently with others.

edited Mar 4 '19 at 8:29

answered Mar 3 '19 at 13:09

Evans AB

41 3

4

Parallel : It's a broad term that means that two pieces of code execute that "at the same time". It doesn't matter if it's "real" parallelism or if it's faked through some clever design pattern. The point is that you can start the "tasks" at the same time and then control them separately (with mutex and all the appropriate tricks). But usually you prefer to use the word "parallel" only for "true" parallelism, as in : you make it happen through non-cooperative multitasking (whether be throuch CPU/GPU cores, or only at software level by letting the OS managing it at a very low level). People are reluctant to say "parallel" just for complicated sequential code that fakes parallelism, like you would find in a browser window's javascript for example. Hence the reason why people in this thread say "asynchronous has nothing to do with parallelism". Well it does, but just don't confuse them.

Concurrent : there can't be concurrency without parallelism (whether simulated or real, as I explained above), but this term focuses specifically on the fact that the two systems will try to access the *same resource* at the same time at some point. It puts the emphasis on the fact that you're going to have to deal with that.

Asynchronous : everyone is right by saying that asynchronous is unrelated with parallelism, but it paves the way to it (the burden is on you to make things parallel or not -- keep reading).

"Asynchronous" refers to a *representation* of parallelism that formalizes the three basic things usually involved in parallelism : 1) define the task's initialization (say when it starts and what parameters it gets), 2) what must be done after it finishes and 3) What the code should continue doing inbetween.

But it's still only syntax (usually it's represented as callback methods). Behind the scene, the underlying system might simply decide that these so-called "tasks" are just fragments of code to pile up until it finishes the code it's currently executing. And then it unpiles them one by one and

executes them sequentially. Or not. It might also create a thread per task and run them in parallel. Who cares? That part is not included in the concept ;)

edited Jun 2 at 11:38

answered Aug 22 '18 at 13:43

jeancallisti

321 2 13

3

I'm going to make it short and interesting to wrap your head around these concepts.

Concurrent vs. Parallel - Ways tasks are executed.

> Take an example in real life: There's a challenge that requires you to both eat a whole huge cake and sing a whole song. You'll win if you're the fastest who sings the whole song and finishes the cake. So the rule is that you sing and eat **concurrently**. How you do that does not belong to the rule. You can eat the whole cake, then sing the whole song, or you can eat half a cake, then sing half a song, then do that again, etc.
>
> **Parallelism** is a specific kind of concurrency where tasks are really executed simultaneously. In computer science, parallelism can only be achieved in multicore environments.

Synchronous vs. Asynchronous - Programming models.

> In sync, you write code as steps that are executed in order, from top to bottom. In an async programming model, you write code as tasks, which are then executed concurrently. Executing concurrently means that all the tasks are likely executed at the same time.

answered Jun 30 at 12:20

Srikanth Bandaru

2,395 2 16 22

3

Here I explain with some examples

Parallelism

A **GPU** uses parallel processing to process *the same block of code* (AKA *kernel*) on thousands of physical and logical threads. Ideally, the process starts and ends for all threads at the same time. A single CPU core without hyperthreading cannot do parallel processing.

Note: *I said ideally because when you run a kernel with size of 7M calls on a hardware with 6M threads, it has to run twice running the same code on all 6M threads in parallel while consuming*

all 6M threads in each time.

- **one kernel** (a piece of code) is executed on multiple processors
- **simultaneously**
- with **a single execution sequence** (a *kernel* must do the same thing in all threads, so "branching" or "if"s are avoided because they will consume the resources drastically by creating lots of NOPs (no-operations) to synchronize all threads)
- essentially **increases speed** drastically
- drastically **limits what you can do**
- highly depends on hardware

Note: *Parallelism is not limited to GPU.*

Concurrency

A **web service** receives *many small requests* in real-time and it needs to handle each of these requests differently, at any time, and independent of other requests or any internal jobs. Yet you want the web service to be up and running at all times without corrupting the state of data or the system health.

Just imagine a user updating a record and another user deleting the same record at the same time.

- **many tasks** are executed
- in **real-time** (or whenever a request comes)
- with **different execution sequences** (unlike kernel in parallel processing, concurrent tasks can do different things, you most likely have to queue or prioritize them)
- essentially **improves the average response time** because task #2 doesn't have to wait for task #1 to finish
- essentially **sacrifices the computational time** because many tasks are running at the same time and there are limited resources
- needs to properly **manage shared resources** so it doesn't run into deadlocks or corrupts the data.

Note*: These requests usually consume some essential resources such as memory, database connection or bandwidth. Yet you want the web service to be responsive at all times.* ***Asynchronousy*** *is the key to make it* ***responsive****, not* ***concurrency***

Asynchronous

One heavy process (like an I/O operation) can easily *block GUI* (or other essential threads) if it's run on the GUI thread. In order to guarantee *UI responsiveness*, a heavy process can be executed asynchronously. It is better to run similar async operations *one at a time*. e.g. multiple IO-bound operations can be significantly slower if run at the same time, so it's better to *queue* them finish to start

- **one task** or **a batch of tasks** is executed on another thread
- **one-time**
- if there is one task, then there is no sequence so you either wait for it to finish or you fire-and-forget
- if it's a batch of tasks then you either fire-and-forget all at the same time, wait for all to finish, or run each task finish to start
- essentially **reduces performance** because of the overheads
- **provides responsiveness to another thread** (effective against blocking of the UI thread or other essential threads)

Note: *an async operation which is executed concurrently (i.e. more than once at a time) is a concurrent operation.*

Note: *Concurrency and asynchronousy are often confused with each-other. Concurrency refers to different parts of the system working together without interfering with each-other (these problems are often solved with locks, semaphors or mutexes). Asynchronousy is how you achieve responsiveness (such as threading).*

***Note**: Asynchronousy and Multithreading are often confused with each-other. Asynchronous code is not necessarily involves a new thread. it can be hardware operation or as Stephan calls it a pure operation, read this

e.g. in the WPF+C# code below, `await Task.Run(()=> HeavyMethod(txt))` is solving an asynchronousy problem, while `textBox.Dispatcher.Invoke` is solving a concurrency problem:

```
private async void ButtonClick(object sender, RoutedEventArgs e)
{
    // run a method in another thread
    await Task.Run(()=> HeavyMethod(txt));

    // modify UI object in UI thread
    txt.Text = "done";
}

// This is a thread-safe method. You can run it in any thread
internal void HeavyMethod(TextBox textBox)
{
    while (stillWorking)
    {
        // use Dispatcher to safely invoke UI operations
        textBox.Dispatcher.Invoke(() =>
        {
```

```
            // UI operations outside of invoke will cause ThreadException
            textBox.Text += ".";
        });
    }
}
```

edited Sep 27 at 11:18

answered Sep 25 '18 at 9:21

Bizhan
12.7k 8 46 77

2

CONCURRENCY VS PARALLELISM: concurrency at one point of time only one task can be done. example: single cpu processor parallelism at one point we can do multiple tasks. example: dual core or multi core processor

answered May 29 '17 at 23:23

rva
21 1