# DOWNLOAD PDF ORIGIN OF CONCURRENT PROGRAMMING

## Chapter 1 : Help Online - License - Concurrent License

*An essential reader containing 19 important papers on the invention and early development of concurrent programming and its relevance to computer science and computer engineering.*

Parallel computing This section has multiple issues. Please help improve it or discuss these issues on the talk page. This section needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. December This section possibly contains original research. Please improve it by verifying the claims made and adding inline citations. Statements consisting only of original research should be removed. December Learn how and when to remove this template message The concept of concurrent computing is frequently confused with the related but distinct concept of parallel computing , [2] [3] although both can be described as "multiple processes executing during the same period of time". In parallel computing, execution occurs at the same physical instant: The goal here is to model processes in the outside world that happen concurrently, such as multiple clients accessing a server at the same time. Structuring software systems as composed of multiple concurrent, communicating parts can be useful for tackling complexity, regardless of whether the parts can be executed in parallel. In this way, multiple processes are part-way through execution at a single instant, but only one process is being executed at that instant. In general, however, the languages, tools, and techniques for parallel programming might not be suitable for concurrent programming, and vice versa. For example, given two tasks, T1 and T2: A set of tasks that can be scheduled serially is serializable , which simplifies concurrency control. For example, consider the following algorithm to make withdrawals from a checking account represented by the shared resource balance: However, since both processes perform their withdrawals, the total amount withdrawn will end up being more than the original balance. These sorts of problems with shared resources need the use of concurrency control , or non-blocking algorithms. Please help improve this section by adding citations to reliable sources. December Learn how and when to remove this template message Concurrent computing has the following advantages: Concurrent programming allows the time that would be spent waiting to be used for another task.

## Chapter 2 : The Origin of Concurrent Programming - - acheter English books | calendrierdelascience.com

*One cannot build or understand a modern operating system unless one knows the principles of concurrent programming. This volume is a collection of 19 original papers on the invention and origins of concurrent programming, illustrating the major breakthroughs in the field from the mid s to the.*

These are three separate actions, and there is no guarantee that no other thread will access the variable until all three are done. But in other possible orderings e. Atomic operations appear to execute as a single machine instruction because all other threads are forced to pause executing while the atomic operation executes. As a result, it is impossible for another thread to observe the value of the updated variables while the operation is in progress. A block of code that requires atomic execution is called a critical section. The critical section mechanism works well in the context of running multi-threaded programs on a computer with a single processor a uniprocessor since it reduces to ensuring that a critical section is not interruptible permitting another thread to run. But it is clumsy and inefficient on a multiprocessor because it forces all processors but one to stop execution for the duration of a critical section. Existing virtual machines treat new operations that force garbage collection as critical sections. A much better mechanism for preventing interference in concurrent programs that may be executed on multiprocessors is locking data objects. When a data object is locked by a thread, no other thread can access or modify the data object until the locking thread releases it. In essence, locking relaxes the concept of atomic execution so that it is relative to a particular object. Threads can continue executing until they try to access a locked object. Java relies on object locking to prevent interference. An object can be locked for the duration of a method invocation simply by prefixing the method declaration with the work synchronized. For instance, to define a synchronized increment method, we would write: Methods that are not declared as synchronized will be executed even when an object is locked! There is a strong argument for this capability: But it also invites subtle synchronization bugs if the synchronized modifier is inadvertently omitted from one method definition. Of course, concurrency only arises in Java when a program uses more than one thread. To support the explicit creation of new threads, Java includes a built-in abstract class Thread, that has an abstract method run. We can make a new thread by i defining a class extending Thread that defines the method run , ii constructing a new instance of this class, and iii calling the start method on this new instance. The start method actually creates a new thread corresponding to the receiver object a Thread and invokes the run method of that thread, much as the main method is invoked in the root class when you run a Java Virtual Machine. At some point in the execution of the original thread now running concurrently with thread t can wait for thread t to terminate by executing the method invocation: So we can view the relationship of the two threads of control as follows: For example, consider the following Java code: Afterwards, all are synchronized and a final value is determined. The counter is not locked in this example, and so updates may be lost because of the problems described above. The likelihood with which update losses may occur varies depending on the number of threads. For example, in a test that I ran a few months ago for 1 million iterations, the program 65 for , iterations, the program lost none. Apparently, even with , threads, each iteration occurred within a single time slice. Synchronizing the threads fixes the problem of lost updates, but it really slows the program down; even for , iterations. In modern event-handling models such as those in Java and DrScheme, we have a single event handler that executes events serially. This protocol saves the overhead of synchronization and eliminates potential deadlocks which we will discuss later.

# Chapter 3 : Concurrency (computer science) - Wikipedia

*This is the opening chapter of the author's book on concurrent programming. The essay describes the fundamental principles of programming which guided the design and implementation of the programming language Concurrent Pascal and the model operating.*

In many fields, the words parallel and concurrent are synonyms; not so in programming, where they are used to describe fundamentally different concepts. A parallel program is one that uses a multiplicity of computational hardware e. Different parts of the computation are delegated to different processors that execute at the same time in parallel , so that results may be delivered earlier than if the computation had been performed sequentially. In contrast, concurrency is a program-structuring technique in which there are multiple threads of control. Notionally the threads of control execute "at the same time"; that is, the user sees their effects interleaved. Whether they actually execute at the same time or not is an implementation detail; a concurrent program can execute on a single processor through interleaved execution, or on multiple physical processors. I recommend reading the rest in the tutorial p. While parallel programming is concerned only with efficiency, concurrent programming is concerned with structuring a program that needs to interact with multiple independent external agents for example the user, a database server, and some external clients. Concurrency allows such programs to be modular; the thread that interacts with the user is distinct from the thread that talks to the database. In the absence of concurrency, such programs have to be written with event loops and callbacks indeed, event loops and callbacks are often used even when concurrency is available, because in many languages concurrency is either too expensive, or too difficult, to use. The notion of "threads of control" does not make sense in a purely functional program, because there are no effects to observe, and the evaluation order is irrelevant. So concurrency is a structuring technique for effectful code; in Haskell, that means code in the IO monad. A related distinction is between deterministic and nondeterministic programming models. A deterministic programming model is one in which each program can give only one result, whereas a nondeterministic programming model admits programs that may have different results, depending on some aspect of the execution. Concurrent programming models are necessarily nondeterministic, because they must interact with external agents that cause events at unpredictable times. Nondeterminism has some notable drawbacks, however: For parallel programming we would like to use deterministic programming models if at all possible. Since the goal is just to arrive at the answer more quickly, we would rather not make our program harder to debug in the process. Deterministic parallel programming is the best of both worlds: Indeed, most computer processors themselves implement deterministic parallelism in the form of pipelining and multiple execution units. While it is possible to do parallel programming using concurrency, that is often a poor choice, because concurrency sacriffices determinism. In Haskell, the parallel programming models are deterministic. However, it is important to note that deterministic programming models are not sufficient to express all kinds of parallel algorithms; there are algorithms that depend on internal nondeterminism, particularly problems that involve searching a solution space. In Haskell, this class of algorithms is expressible only using concurrency.

# Chapter 4 : Per Brinch Hansen: Origin of Concurrent Programming (PDF) - ebook download - english

*'Origin of Concurrent Programming' by Per Brinch Hansen is a digital PDF ebook for direct download to PC, Mac, Notebook, Tablet, iPad, iPhone, Smartphone, eReader - but not for Kindle.*

# Chapter 5 : The Origin of Concurrent Programming - calendrierdelascience.com Hansen - Librairie Eyrolles

*Edited by a computer pioneer, "The Origin of Concurrent Programming" is an essential reader on the historic development of concurrent programming. It is an invaluable resource for students, researchers and professionals who are familiar with operating system principles.*

# Chapter 6 : What is Concurrent Programming?

*The author selects classic papers written by the computer scientists who made the major breakthroughs in concurrent programming. These papers cover the pioneering era of the field from the semaphores of the mid s to the remote procedure calls of the late s.*

# Chapter 7 : Help Online - License - FLEXnet Server Setup for Windows

*Concurrent computation makes programming much more complex. In this section, we will explore the extra problems posed by concurrency and outline some strategies for managing them. In a concurrent program, several streams of operations may execute concurrently.*

# Chapter 8 : The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls - Goog

*Concurrent computing is a form of computing in which several computations are executed during overlapping time periodsâ€"concurrentlyâ€"instead of sequentially (one completing before the next starts).*

# Chapter 9 : Concurrent computing - Wikipedia

*While parallel programming is concerned only with efficiency, concurrent programming is concerned with structuring a program that needs to interact with multiple independent external agents (for example the user, a database server, and some external clients).*