Lecture 4: Threads & Concurrency

Adam Hawley

January 24, 2019

Contents

1 Concurrency

1.1 Multiple Processes

Applications are often constructed from multiple co-operating programs.

For example, web browsers used to run as a single process, therefore if oone site causes trouble, the entire browser can hang or crash. Google Chrome is multi-process with 3 different types of processes:

- Browser process manages user interface, disk and network I/O
- Renderer process renders web pages, deals with HTML and js (with a new renderer created for each open website)
- Each type of plugin has a separate process

1.2 Motivation

- **Responsiveness** — program may continue execution even if a process if blocked, especially important for user interfaces.
- **Modularity** — it may be easier to create a program as a set of interacting processes.
- **Scalability** — process can take advantage of multiprocesssor architectures.

1.3 Multiprocessor Systems

Multi-CPU Systems — Multiple CPUs are placed in the computer to provide more computing performance.

Multicore Systems — Multiple computing cores are places on a single processing chip where each core appears as a separate CPU to the operating system.

Concurrent programming provides a mechanism for more efficient use of these multiple computing cores and improved concurrency. On a system with a sigle computing core, concurrency means that the execution of the processes will be interleaved over time. But on a system with multiplr cores, concurrency means that some processes can run in parallel, because the system can assign a separate process to each core.

1.4 Concurrency vs. Parallelism

There is a fine but clear distinction between concurrency and parallelism. A **concurrent** system supports more than one task by allowing all the tasks to make progress. In contrast, a system is **parallel** if it can perform more than one task simultaneously. Thus, it is possible to have concurrency without parallelism. Concurrency within a program is an opportunity for parallelism if there is parallel hardware to exploit.

Types of parallelism:

- Data parallelism — distributes subsets of the same data across multiple cores, same operation on each.
- Task parallellism — distributing threads across cores, each thread performing unique operation.

1.5 Amdahl's Law

Amdahl's law indentifies performance gains from adding additional cores to an application that has both serial and parallel components:

Where N is the number of processing cores and S is the serial portion:

$$speedup \leq \frac{1}{S + \frac{1-S}{N}}$$

That is, if an application is 75% parallel and 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times. As N approaches infinity, speedup approaches $\frac{1}{S}$. As S approaches 0, speedup approaches N. Serial portion of an application has disproportionate effect on performance gained by adding additional cores.

2 Threads

2.1 Intro

Multiple concurrent processes are conceptually equivalent to multiple independent programs (each has a separate address space and requires inter-process communications). **Threads** are concurrent units within a process. All threads of a process share the same address space, low overhead in thread creation. Benefits from shared-memory processors, low-inter thread communication overhead.

2.2 User & Kernel Threads

Support for threads may be provided at two different levels:

- **User threads** — are supported above the kernel and are managed without kernel support, primarily by user-level threads library.
- **Kernel threads** — are supported and managed directly by the operating system.

Virtually all contempory systems support kernel threads.

2.2.1 Many-to-One Relationship

Many user-level threads mapped to single kernel thread. One thread blocking causes all to block. Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time. Few systems still use this model.

2.2.2 One-to-One Relationship

Each user-level thread maps to a single kernel thread. Creating a user-level thread creates a kernel thread. More concurrency than many-to-one. Number of threads per process sometimes restricted due to overhead. Examples:

- Windows 95-XP
- Linux

2.2.3 Many-to-Many Relationship

Allows many user level threads to be mapped to many kernel threads. Allows the operating system to create a sufficient number of kernel threads. Examples:

- Solaris prior to version 9
- Windows with the *ThreadFiber* package

2.3 Thread Libraries

Thread library provides programmer with an API for creating and managing threads. Two primary ways of implementing:

- Library entirely in user space
- Kernel-level library supported by the OS

Widely used thread libraries:

- POSIX Pthreads
- Windows threads
- Java threads

3 Threading Issues

3.1 Thread Cancellation

Thread cancellation means terminating a thread before it has finished working. There can be two approaches to this:

- **Asynchronous cancellation** — terminates the target thread immediately. (Immediate and responsive but more dangerous)
- **Deferred cancellation** — allows the target thread to periodically check if it should be cancelled.

3.2 Signal Handling

Signals are used in UNIX systms to notify a process that a particular event has occured. When a multithreaded process receives a signal, OS can be set to deliver it to all or to specific thread(s).

3.3 fork() System Call

If one thread of a process calls `fork()`, will the entire process be duplicated, or only the calling thread?

Some versions of UNIX have two versions of `fork()` and the choice between them often depends on whether `exec()` will be called afterwards.

`exec()` usually works as normal — replace the running process including all threads.

3.4 Security and Integrity Issues

There are many security and integrity isuues because of the sharing of resources between multiple threads. This will be covered in greater detail later in the module.