Contents

Concurrency: An Introduction

Introduction

thread

Instead of a single point of execution within a program a **multi-threaded** program has more than one point of execution

another way to think of this

each thread is very much like a separate process, except they share the same address space and thus can access the same data.

The state of a thread

is very similar to that of a process. It has a program counter (PC) Each thread has its own private set of registers it uses for computation;

context switch

The context switch between threads is quite similar to the context switch between processes, There is one major difference, the address space remains the same (i.e., there is no need to switch which page table we are using).

Why Use Threads?

parallelism.

if you are executing the program on a system with multiple processors, you have the potential of speeding up this process considerably by using the processors to each perform a portion of the work.

avoid blocking program progress

due to slow I/O. Using threads is a natural way to avoid getting stuck; Threading enables overlap of I/O with other activities within a single program,

The Heart Of The Problem: Uncontrolled Scheduling

the results depend on the timing execution of the code. With some bad luck (i.e., context switches that occur at untimely points in the execution), we get the wrong result.

mutual exclusion.

This property guarantees that if one thread is executing within the critical section, the others will be prevented from doing so.

TIP: USE ATOMIC OPERATIONS

atomic

"all or nothing"; it should either appear as if all of the actions you wish to group together occurred, or that none of them occurred, with no in-between state visible.

Sometimes,

Sometimes, the grouping of many actions into a single atomic action is called a transaction,

The Wish For Atomicity

ask the hardware for a few useful instructions upon which we can build a general set of what we call **synchronization primitives.** By using this hardware support, in combination with some help from the operating system, we will be able to build multi-threaded code that accesses critical sections in a synchronized and controlled manner,

One More Problem: Waiting For Another

ASIDE: KEY CONCURRENCY TERMS

critical section

is a piece of code that accesses a shared resource, usually a variable or data structure.

race condition

arises if multiple threads of execution enter the critical section at roughly the same time; both attempt to update the shared data structure, leading to a surprising outcome.

indeterminate program

consists of one or more race conditions; the output of the program varies from run to run, depending on which threads ran when. The outcome is thus not **deterministic,**