Nguyen Phan Yen Ngan – 14597

Chapter 4: Threads and Concurrency – Operating System Concepts

4.1 Overview

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack.

4.1.1 Motivation

-Software applications that run on modern computers and mobile devices are multithreaded.

- Applications can be designed to leverage processing capabilities on multicore systems. They can perform several CPU-intensive tasks in parallel across the multiple computing cores.

- In some situations, a single application may be required to perform several similar tasks. One solution is to have the server run as a single process that accepts requests. When the server receives a request, it creates a separate process to service that request.

- Most operating system kernels are also typically multithreaded.

- Many applications can also take advantage of multiple threads, including basic sorting, trees, and graph algorithms.

4.1.2 Benefits

- Responsiveness

- Resource sharing

- Economy

- Scalability

4.2 Multicore Programming

4.2.1 Programming challenges

- Identifying tasks

- Balance

- Data splitting

- Data dependency

- Testing and debugging

4.2.2 Types of parallelism

- Data parallelism: focuses on distributing subsets of the same data across multiple computing cores and performing the same operation on each core.

- Task parallelism: involves distributing both data and tasks (threads) across multiple computing cores. Each thread is performing a unique operation.

4.3 Multithreading model

4.3.1 Many-to-one model

- It maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient.

4.3.2 One-to-one model

- It maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call.

4.3.3 Many-to-many model

- It multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine. Whereas the manyto-one model allows the developer to create as many user threads as she wishes, it does not result in parallelism, because the kernel can schedule only one kernel thread at a time.

- One variation on the many-to-many model still multiplexes many userlevel threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread. This variation is sometimes referred to as the two-level model.

4.4 Thread libraries

- A thread library provides the programmer with an API for creating and managing threads. There are two primary ways of implementing a thread library. The first approach is to provide a library entirely in user space with no kernel support. The second approach is to implement a kernel-level library supported directly by the operating system.

4.1 Pthreads

- It refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation and synchronization. This is a specification for thread behavior, not an implementation.

4.2. Windows

- The technique for creating threads using the Windows thread library is similar to the Pthreads technique in several ways. We illustrate the Windows thread API in the C program.

4.3. Java

- Threads are the fundamental model of program execution in a Java program, and the Java language and its API provide a rich set of features for the creation and management of threads.

- All Java programs comprise at least a single thread of control.

- Java threads are available on any system that provides a JVM including Windows, Linux, and macOS. The Java thread API is available for Android applications as well.

5. Implicit threading

5.1. Thread pools

- The first issue concerns the amount of time required to create the thread, together with the fact that the thread will be discarded once it has completed its work.

- The second issue is more troublesome. If we allow each concurrent request to be serviced in a new thread, we have not placed a bound on the number of threads concurrently active in the system.

- Unlimited threads could exhaust system resources, such as CPU time or memory. One solution to this problem is to use a thread pool.

5.2. Fork join

- Recall that with this method, the main parent thread creates (forks) one or more child threads and then waits for the children to terminate and join with it, at which point it can retrieve and combine their results.

5.3. OpenMP

- OpenMP identifies parallel regions as blocks of code that may run in parallel.

5.4. Grand central dispatch.

- It is a combination of a run-time library, an API, and language extensions that allow developers to identify sections of code (tasks) to run in parallel.

- GCD schedules tasks for run-time execution by placing them on a dispatch queue. GCD identifies two types of dispatch queues: serial and concurrent.

- Tasks placed on a serial queue are removed in FIFO order. Once a task has been removed from the queue, it must complete execution before another task is removed. Each process has its own serial queue (known as its main queue).

5.5. Intel Thread Building Blocks

- A template library that supports designing parallel applications in C++. As this is a library, it requires no special compiler or language support.

6. Threading Issues

- The fork() and exec() System Calls

- Signal Handling

- Thread Cancellation

- Thread-Local Storage

- Scheduler Activations