Summary on multithreading

by Arthur Golubev 19850316

2025-07-06

Table of Contents

1: Threads

Every program is a set of commands which also can be called instructions; for example, to calculate something, to signal to a device, to read from a channel. Some actions cannot happen simultaneously; for example, when we want to calculate something and then send it to an output device, calculating must be completed before sending this result to an output device; and some actions can happen simultaneously, for example providing result of previous calculations to an output device and performing next calculations. Also, what actions can be performed simultaneously depends on what apparatuses can perform simultaneously.

Program commands can be organized into set of sequences of commands that depend on each other only by shared data; such sequences are called threads. Some resources and information can be associated with a thread; for examples, thread memory or security access information, respectively.

Number of threads that an apparatus can perform simultaneously may be less than occasionally is current number of threads of a program, then some dispatching of thread over apparatus resources is required; replacing a thread to switch apparatus resources to another thread can be done:

- by a thread itself (a thread itself calls commands of switching to another thread);

- forcedly to a thread: by either hardware or operating system.

2: Things to consider on multithreading

2.1: Keeping correctness of data when multithreading

2.1.1: Problem of nonatomicity of some operations

Atomicity of an operation in a system means that the operation in the system cannot be broken into parts.

If a threads reads or modifies shared data by nonatomic operations simultaneously with another thread is modifying the shared data, these operations can be performed with corrupting values.

An example of such corruption of value because of absence of atomicity follows. In this example of corruption of values because of absence of atomicity will be used term byte; word byte means a portion of information that has its own address in a system.

Imagine that a program needs to operate a variable number of diverse values of which is more than number of diverse values of a byte. Lets consider in the system the possible number of values in a byte is 256 an we need to operate a piece of data which fits in minimum two bytes. Common way to store such piece of data in two such bytes is storing units in the one of the byte with weight 256 (so that contribution of this byte is its value * 256) and units the another with weight 1 (so that contribution of this byte is 1). Lets consider we decided that units of the first byte weights 256 and units of the second byte weights 1. Lets consider and we want the initial value is 515; then in the first byte would be placed 2 and in the second byte would be placed 3 (the total contribution would be 256 * 2 + 1 * 3 = 515). Lets consider a situation that one of threads is reading the bytes and another is changing the value to a 0 so that after the first threads has read the first byte, the second thread changed both bytes to 0. Completing the operations the first thread reads the second byte so that for it the bytes values will be 0 and 3 which means value 256 * 0 + 3 * 1 = 3 which is neither initial, 515, nor set by the second of the threads 0 value.

2.1.2: Avoiding inappropriate accesses to shared data

Some operations are naturally atomic according to design of the system; for examples:

- as apparatus constructed changing apparatus flag is performed an operation per apparatus tick so that next operation just cannot start until the previous operation has completed because the next tick has not yet started;

- as apparatus constructed commands of changing state in a device are transferred via a single bus to the device so that can pass only one by one.

On level of apparatuses, lets call memory atomic for the operations if these operations for this memory are atomic. On level of programs, lets call variables atomic for the operations if these operations for this variable are atomic.

Variable that indicates some state is called flag.

If it is appropriate to the system, program can just store a value it operates by threads in atomic memory.

Another option is using an atomic flag per shared data to indicate that one of operations which we want to not be performed simultaneously either is going to start or has completed; then threads can check the atomic flag and decide whether to start their operations on the shared data or wait until a concurrent operation has completed. It is an option how many data protect with a flag; it designed has to be considered how expensive working with flags is, how expensive storing flags is and how expensive changing data structure is.

In case of access to the atomic memory happens through cache memory either automatic or by commands managing of correct updating the atomic memory and caches is required.

There is a trick when data is accessed through a pointer, to protect access by an atomic flag only for the pointer and rather than update data create a new instance and then redirect the pointer to the new instance.

2.2: Thread pools

When apparatus or operating system switches what thread it is running it costs and when a program creates and deletes information associated with threads it costs. It may happen to be cheaper to create threads which dispatch and perform pieces of code of tasks rather than to dispatch threads for tasks and create and delete information associated with the threads for tasks.