Multithreading

Introduction

Multithreading is a concept where a program is broken into two or more parts called threads and all these threads run in parallel. Multithreading can make programs more responsive and effective and it increases its performance too. For example, today web pages need to display animations with sound effects and text at the same time. If this is done by using the traditional single-threaded event loop, the application will take a lot of time to get loaded, and time is a very crucial factor for web applications.

While running a program without using multithreading, it follows an approach called an event loop with polling. In this approach, a single process takes the control and decides what is to be done next. It uses polling with a single event queue mechanism to decide this. Once the polling mechanism returns with a signal that a particular resource is available, it dispatches the request to the appropriate event handler. Until this event handler returns nothing else can happen. Let us take an example to understand this. Let us suppose that a user, while working in a spreadsheet package, wants to calculate a complex formula and also wants to scroll down. Both of these requests go in the event queue one by one. The first request, i.e., the calculation of these requests go in the queue first. Let us assume that the formula is very complex and takes a lot of time to get calculated. In such a case, the second event will keep asking for the resource, but the polling mechanism will return with a signal that the resource is not free. Therefore, the system would wait till the formula gets calculated and then only the scrolling will start.

Earlier languages like C and C++ did not support multithreading since most of the operating systems at that point of time did not support it. Operating systems like Microsoft Windows, Apple Macintosh, etc. were designed around traditional event loop model.

Why Multithreading?

Multithreading exploits the fact that most of the time the tasks (parts) of the same program are either waiting for the other resources to become free, or waiting for some timeout to occur. In the above example (spreadsheet), scroll operation is waiting for the calculation to be completed. If these parts or tasks can be described as independent threads, they can run individually and they don’t have to wait for the other threads to be completed. One can automatically switch from one task that is ready to wait to another task that is ready. In multithreading, two different tasks – performs calculations and scrolling, form two different threads. When the first

thread performs calculation, the second thread can make the window scroll down. This would give the notion of both the operation being performed simultaneously to the user and improves the effectiveness of the application.

Multithreading can be done in a single-processor or multiple processor environments. In a multiple processor environment, different threads can run on different processors.

In a single-processor system, however the things happen in a different way. In such a system, multiple threads share the CPU time. The operating system is responsible for scheduling and allocating resources to threads. This arrangement is very practical as a thread does not always need the CPU time. It might have to wait for user input or it might have to display something on the screen. During that time, the other thread takes over the CPU. The previous thread can then resume after the current thread has either utilized all CPU time or the thread has ended.

Source: http://www.go4expert.com/articles/multithreading-t4174/