2016

International Journal of Humanities & Information Technology (IJHIT)

Volume – 1, Issue - 1

ISSN : Applied

INTERNATIONAL JOURNAL OF HUMANITIES & INFORMATION TECHNOLOGY (IJHIT)

WWW.IJHIT.COM

International Journal of Humanities & Information Technology (IJHIT)
*e-ISSN: **Applied**, Volume 1, Issue 1, (January 2016), www.ijhit.com*

Use of Core System in Multitasking in Parallel Computing

Sanju Pillai[1], Rameshwar D. Pillai[2]

[1,2](Associate Professor/Department of Computer Science/ Centre for Development Studies/ Thiruvananthpuram)

Abstract: *In parallel computing, multitasking is a method by which multiple tasks, also known as processes, share common processing resources such as a CPU. With a multitasking OS, such as Windows XP, you can simultaneously run multiple applications. Multitasking refers to the ability of the OS to quickly switch between each computing task to give the impression the different applications are executing multiple actions simultaneously.*
As CPU clock speeds have increased steadily over time, not only do applications run faster, but OSs can switch between applications more quickly. This provides better overall performance. Many actions can happen at once on a computer, and individual applications can run faster.

Keywords: *Time sharing, Multicore, Context switching, CPU clock speed*

1. Introduction

Single core processor can run a single process (task) at a time. Only one thread can execute at a time but the Operating system achieves Multithreading using time slicing (thread context switch). This Thread switching happens frequently enough that the user perceives the threads as running at the same time (but they aren't running parallel!) and it occurs inside the one process.

A Process context switch is similar to thread context switch with a difference that it takes place between processes (example between media player und notepad) instead between threads.

I'm not sure if this example is valid : taking two processes e.g.: Notepad and Media player on a single core processor. One can play music and write in a Notepad at the same time although the two processes aren't running parallel (Process context switching or multitasking).Inside the one process e.g. : Media player one can listen to music and create playlists at the same time although the two threads aren't running parallel (Thread context switch or multithreading). There are two type of core system in multitasking system of parallel computing.

1.1. Single Core

In the case of a computer with a single CPU core, only one task runs at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves this problem by scheduling which task may run at any given time and when another waiting task gets a turn.

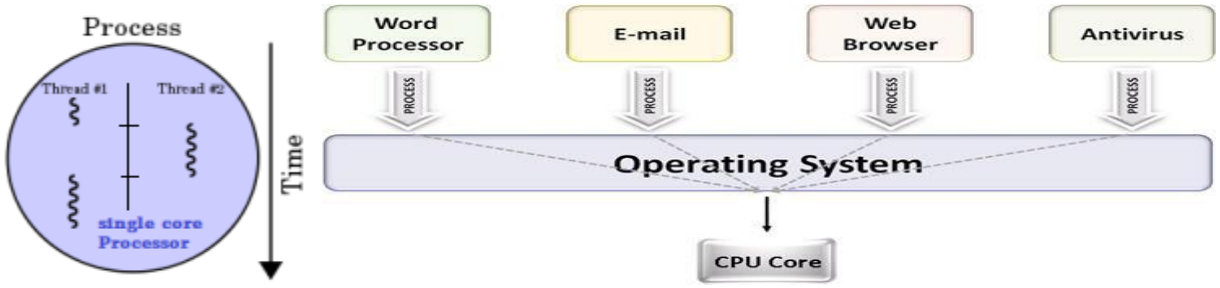

Figure 1. Single-core systems schedule tasks on 1 CPU to multitask

1.2. Multicore

When running on a multicore system, multitasking OSs can truly execute multiple tasks concurrently. The multiple computing engines work independently on different tasks.
For example, on a dual-core system, four applications - such as word processing, e-mail, Web browsing, and antivirus software - can each access a separate processor core at the same time. You can multitask by checking e-mail and typing a letter simultaneously, thus improving overall performance for applications.

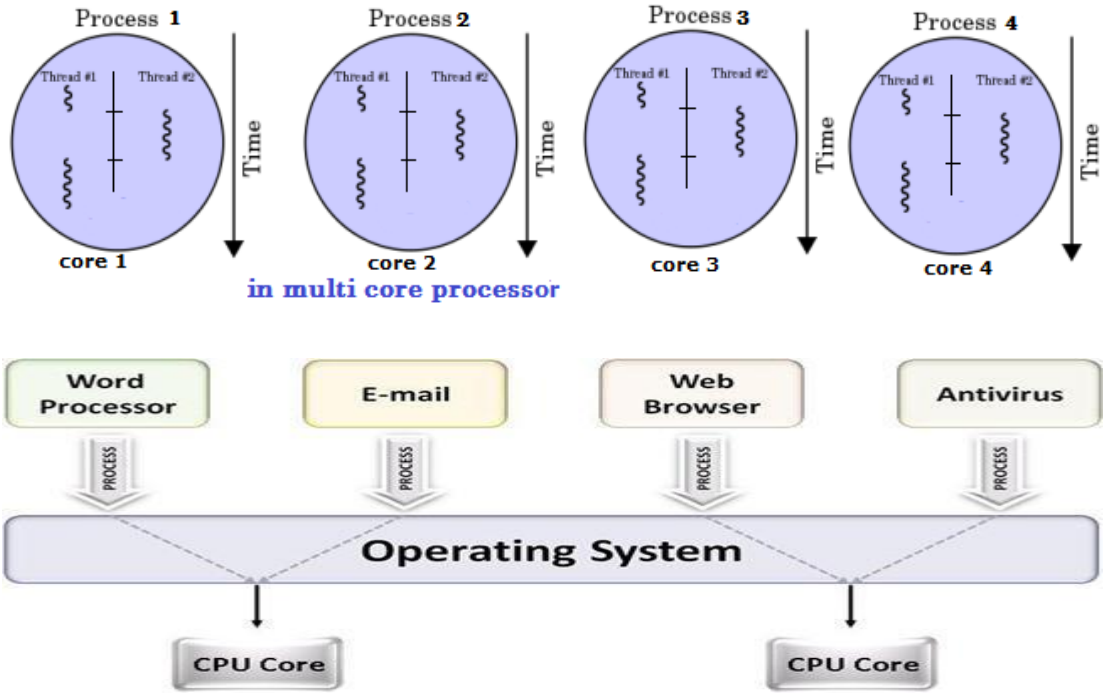

Figure 2. Dual-core systems enable multitasking operating systems to execute two tasks simultaneously

The OS executes multiple applications more efficiently by splitting the different applications, or processes, between the separate CPU cores. The computer can spread the work - each core is managing and switching through half as many applications as before - and deliver better overall throughput and performance. In effect, the applications are running in parallel.

1.3. Multithreading

Multithreading extends the idea of multitasking into applications, so you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

In a multithreaded National Instruments LabVIEW program, an example application might be divided into four threads - a user interface thread, a data acquisition thread, network communication, and a logging thread. You can prioritize each of these so that they operate independently. Thus, in multithreaded applications, multiple tasks can progress in parallel with other applications that are running on the system.

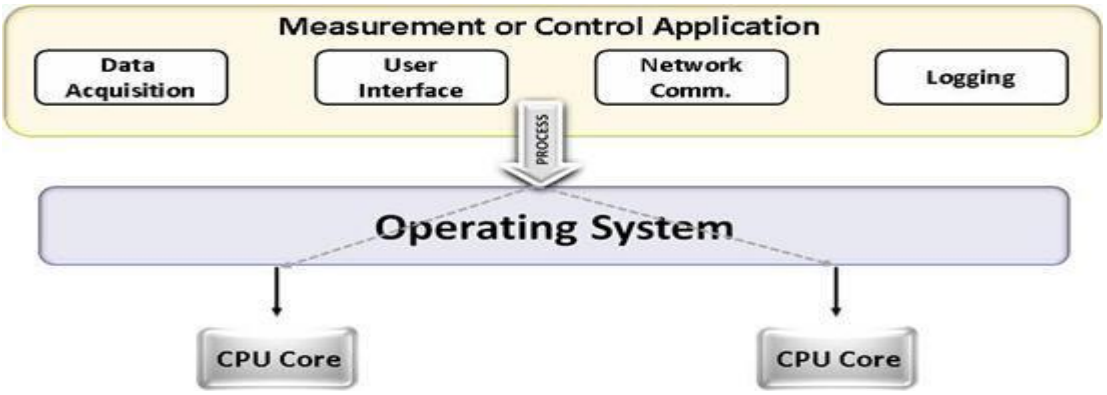

Figure 3. Dual-core system enables multithreading

Applications that take advantage of multithreading have numerous benefits, including the following:

- More efficient CPU use
- Better system reliability
- Improved performance on multiprocessor computers

In many applications, you make synchronous calls to resources, such as instruments. These instrument calls often take a long time to complete. In a single-threaded application, a synchronous call effectively blocks, or prevents, any other task within the application from executing until the operation completes. Multithreading prevents this blocking.

While the synchronous call runs on one thread, other parts of the program that do not depend on this call run on different threads. Execution of the application progresses instead of stalling until the synchronous call completes. In this way, a multithreaded application maximizes the efficiency of the CPU because it does not idle if any thread of the application is ready to run.

1.4. Multithreading with LabVIEW

LabVIEW automatically divides each application into multiple execution threads. The complex tasks of thread management are transparently built into the LabVIEW execution system.

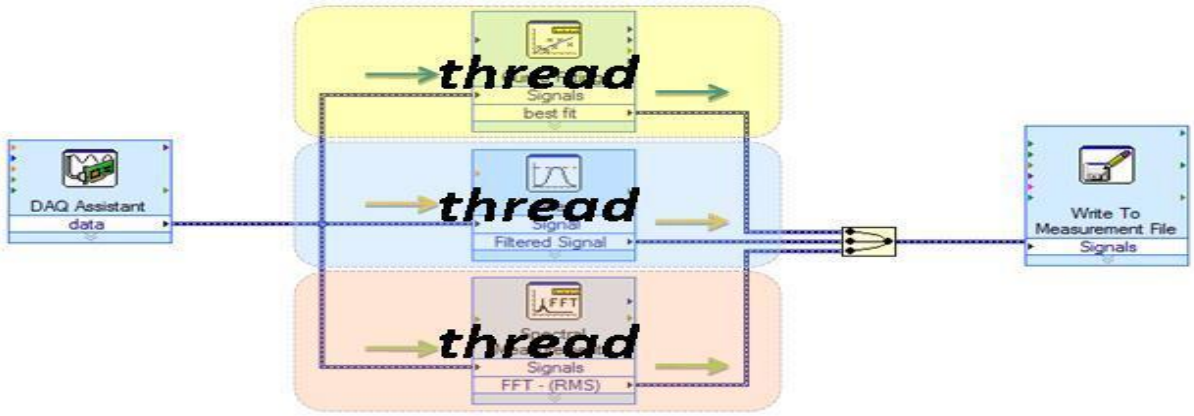

Figure 4. LabVIEW uses multiple execution threads

2. Conclusion

LabVIEW uses preemptive multithreading on OSs that offer this feature. LabVIEW also uses cooperative multithreading. OSs and processors with preemptive multithreading employ a limited number of threads, so in certain cases, these systems return to using cooperative multithreading.

The execution system preemptively multitasks VIs using threads. However, a limited number of threads are available. For highly parallel applications, the execution system uses cooperative multitasking when available threads are busy. Also, the OS handles preemptive multitasking between the application and other tasks.

References

[1]. Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.
[2]. "Concurrency is not Parallelism", Waza conference Jan 11, 2012, Rob Pike (slides) (video)
[3]. "Parallelism vs. Concurrency". Haskell Wiki.
[4]. Hennessy, John L.; Patterson, David A.; Larus, James R. (1999).
[5]. Computer organization and design: the hardware/software interface (2. ed., 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6.
[6]. Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved 2007-11-09.
[7]. Hennessy, John L.; Patterson, David A. (2002). Computer architecture / a quantitative approach. (3rd ed.). San Francisco, Calif.: International Thomson. p. 43. ISBN 1-55860-724-2.
[8]. Rabaey, Jan M. (1996). Digital integrated circuits : a design perspective. Upper Saddle River, N.J.: Prentice-Hall. p. 235. ISBN 0-13-178609-1.