Thread In OSs

23/09/2006 00:00 by the6dmin

Threads implementation in Windows and Linux

Elements that are set to execute within a program are known as threads. Threads enhance overall system performance by allowing concurrent operations to occur during program execution. On multi-processor systems, several instances (threads) can run in parallel on several processors; enabling a program to run faster. Multi-threaded applications have gained popularity over their native single-threaded programs in favor of adding more system functionalities for users.

Threads in Windows

When a program runs, Windows creates a process and threads within to execute the various areas of the program as per the functionality it was developed for. Threads make use of the same memory address space allocated to their parent process in addition to other resources. To identify various threads within a process; each has a thread identifier, a priority scheduler, a local storage and a thread context. Windows creates threads using the "CreateThread" function that returns a handler to identify the thread and allow it to spawn child threads. The number of threads created in Windows is limited by the amount of memory (main and virtual) available on the system. A thread can create other worker threads, also known as a "thread pool", and use public variables to share vital data or pass any necessary stack sizing, arguments and security attributes through the "CreateThread" function parameters. The thread pool is assigned small tasks by the main thread to asynchronously manage work on behalf of the application. The technology is been constantly developed by Microsoft to include waiter threads, a work queue and, a worker factory that manages the worker threads.

Threads are dispatched to the processor for execution; the scheduler code is within the system kernel, there is no pre-defined module or procedure. To change the execution order of a thread; Windows changes the priority of the parent process then sets a new priority for the thread. A thread can suspend or resume another thread, since threads do not run in synchronous; a flag is sent using the "ResumeThread" function call with the suspended thread's handler as a parameter. This represents a key benefit in multi-threading, in that it implements simultaneous processing. When the task is completed the thread calls the function "ExitThread" to terminate, also terminating the parent process will result in terminating any thread belonging to it. The operating system does some house cleaning by freeing any resources allocated to that thread, setting the thread exit code, and terminating the main process if that thread was the only one. A thread can exit through a call to the "TerminateThread " function by another thread, this will not allow any cleaning to be done, leaving resources and objects still in association with the thread. Windows uses preemptive multitasking, in which a thread shares the memory space and voluntarily gives up the processor time allocated to it for another thread to execute. If any thread misbehaved due to errors in programming, a thread can be forced to terminate using the designated terminate function.

[-pagebreak-]

Threads in Linux

Unlike Windows, Linux main execution element is the process. The term thread and process was used interchangeable in Linux. Linux threading has been introduced and gone under extensive development to define its characteristics aside from processes. Earlier, Linux used to mimic a thread by generating a process through the "clone" function; the copy process shared its parent memory address and other resources. This implementation suffered scheduling and inter-process communication problems that raised the need to generate a complete Linux thread module. The LinuxThread library, one of many others including NGPT and NPTL, implements the thread in the user not the kernel space by a call to "pthread_create" function. In addition, the creation involves calls to functions that initialize and set parameters for the stack size, a step that is implicitly included within the "CreateThread " function used by Windows. Linux allows the administrator to modify the maximum number of threads created per user, a setting that is lost when the

operating system reloads. Linux threading module is based on an "on-on-one" model; threads are born joined and can synchronize on other threads termination; making use of their resources which are freed once the join is complete.

A major difference between Windows threads and Linux is the priority independence of Linux threads from their parent processes. While Windows uses function calls among threads to change states, Linux uses signals to, for example, awaken a thread from its block state. Linux maintains a similar set of standards to Windows when it comes to threads termination. The "pthread_exit " function is called by a thread to terminate it leaving the parent process as is. Terminated threads are managed as not to turn into zombies. Only one difference is that Linux uses a manager thread, which acts as a coordinator among other threads within a certain process. This thread never terminates until all other threads are killed and thus freeing all the resources allocated to the main process.

References

Kavitha Varadarajan, "Port Windows IPC apps to Linux, Part 1: Processes and threads," IBM developerWorks, 15 Apr 2005, http://www-128.ibm.com/developerworks/linux/library/l-ipc2lin1.html > (25 Aug 2005). The MSDN Library 2006, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/thread_pools.asp Xavier Leroy, "Linuxthreads - POSIX 1003.1c kernel threads for Linux," LinuxThread ReadMe, 1996, http://pauillac.inria.fr/~xleroy/linuxthreads/README(1997). Sean Walton, "LinuxThreads Frequently Asked Questions, " The Linux Documentation Project , 21 Jan 1997, http://tldp.org/FAQ/Threads-FAQ/index.html > (21 Jan 1997).