Kernel comparison of OpenSolaris, Windows Vista and Linux 2.6

The idea of writing this paper is evoked by Max Bruning's view on Solaris, BSD and Linux. The comparison of advantages and disadvantages among quasi-Unix systems is an oft-told tale. However, this article looks into the three kernel subsystems of the latest operating system release – OpenSolaris, Windows Vista and Linux kernel 2.6. The simple reason is that they are the most widely used and welcomed operating systems within business environment and developer communities.

There are lots of criteria to value a system, but undoubtedly, the fundamental role of an operating system in computer science remains unchanged. It can be thought of as having three objectives:

- ✓ **Efficiency**: allowing system resources (especially hardware) to be used in an efficient manner.

- ✓ **Evolvement**: The only thing that never changes is change. OS should be constructed in such a way as to permit the effective development and introduction of new system functions without interfering with original service.

- ✓ **User-friendliness**: We need to face OS every day. A user-friendly interface is compulsory, otherwise you will be dropped not matter how well you follow the above two criteria.

Inevitability, OS needs functions like process/thread management, which allocates and calls threads under a particular dispatcher policy, memory management and file management. We will compare these subsystems head by head. (User friendliness will not be discussed here in this article, since we are focusing on kernel comparison.) There are some similar concepts between Linux and OpenSolaris, while concepts in Windows Vista are quite different.

Process and Thread Management

OpenSolaris

OpenSolaris implements multilevel thread support designed to provide considerable flexibility in exploiting processor resources. Four following new concepts are implemented in OpenSolaris.

- **Process**: This is the normal UNIX process and includes the user's address space, stack, and process control block.
- **User-level threads**: Implemented through a threads library in the address space of a process, these threads are invisible to the OS.A user-level thread (ULT)10 is a user-created unit of execution within a process.
- **Lightweight processes**: A lightweight process (LWP) can be viewed as a mapping between user level threads and kernel threads. Each LWP supports ULT and maps to one kernel thread. LWPs are scheduled by the kernel independently and may execute in parallel on multiprocessors.
- **Kernel threads**: These are the fundamental entities that can be scheduled and dispatched to run on one of the system processors.

As is showed in Figure 0, *ps -Lec* command display system processes and threads.

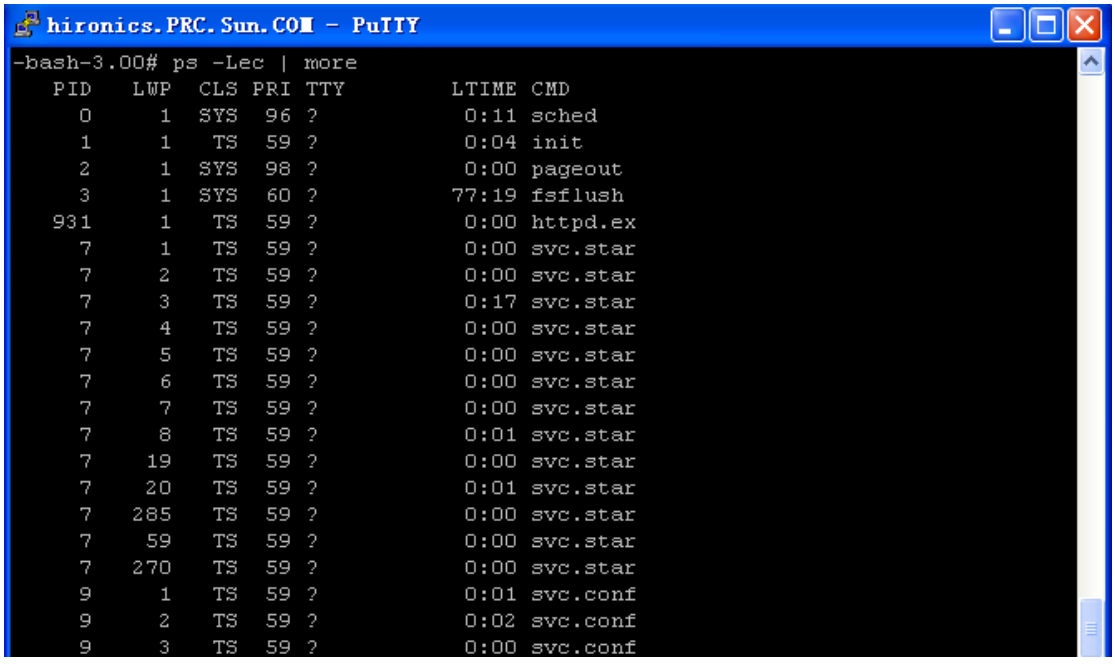

Figure 0 Output from Solaris 10 update 6

Figure 1 below illustrates the relationship among these four entities.

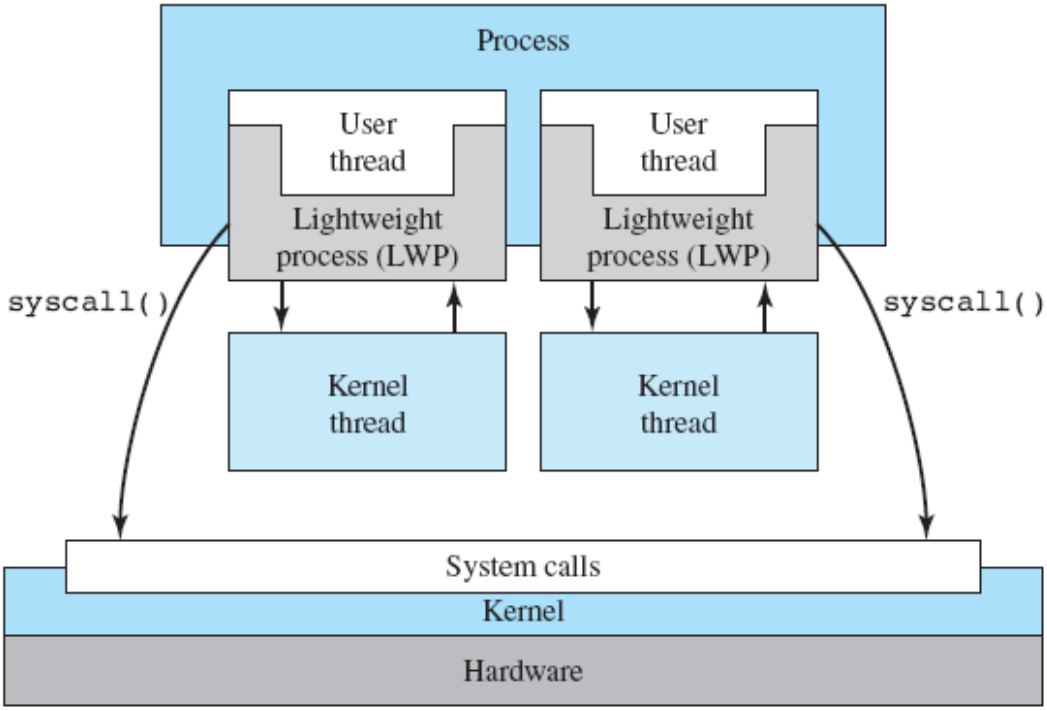

Figure 1 opensolaris thread model[1]

The three-level thread structure (ULT, LWP, kernel thread) in OpenSolaris is intended to facilitate thread management by the OS and to provide a clean interface to applications.

The user thread interface can be a standard thread library. A defined ULT maps onto a LWP, which is managed by the OS and which has defined states of execution, defined subsequently. An LWP is bound to a kernel thread with a one-to-one correspondence in execution states. Thus, concurrency and execution is managed at the level of the kernel thread.

In addition, an application has access to hardware through an application program interface (API) consisting of system calls. The API allows the user to invoke kernel services to perform privileged tasks on behalf of the calling process, such as read or write a file, issue a control command to a device, create a new process or thread, and allocate memory for the process to use, and so on.
The change of thread model drivers the altering of process data structure. OpenSolaris retains this basic structure but replaces the processor state block with a list of structures containing one data block for each LWP.

The LWP data structure includes the following elements:

An LWP identifier

- The priority of this LWP and hence the kernel thread that supports it

[1] Richard McDougall, Jim Mauro ,Solaris internals 2005 Pearson Education

- A signal mask that tells the kernel which signals will be accepted
- Saved values of user-level registers (when the LWP is not running)
- The kernel stack for this LWP, which includes system call arguments, results, and error codes for each call level
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure

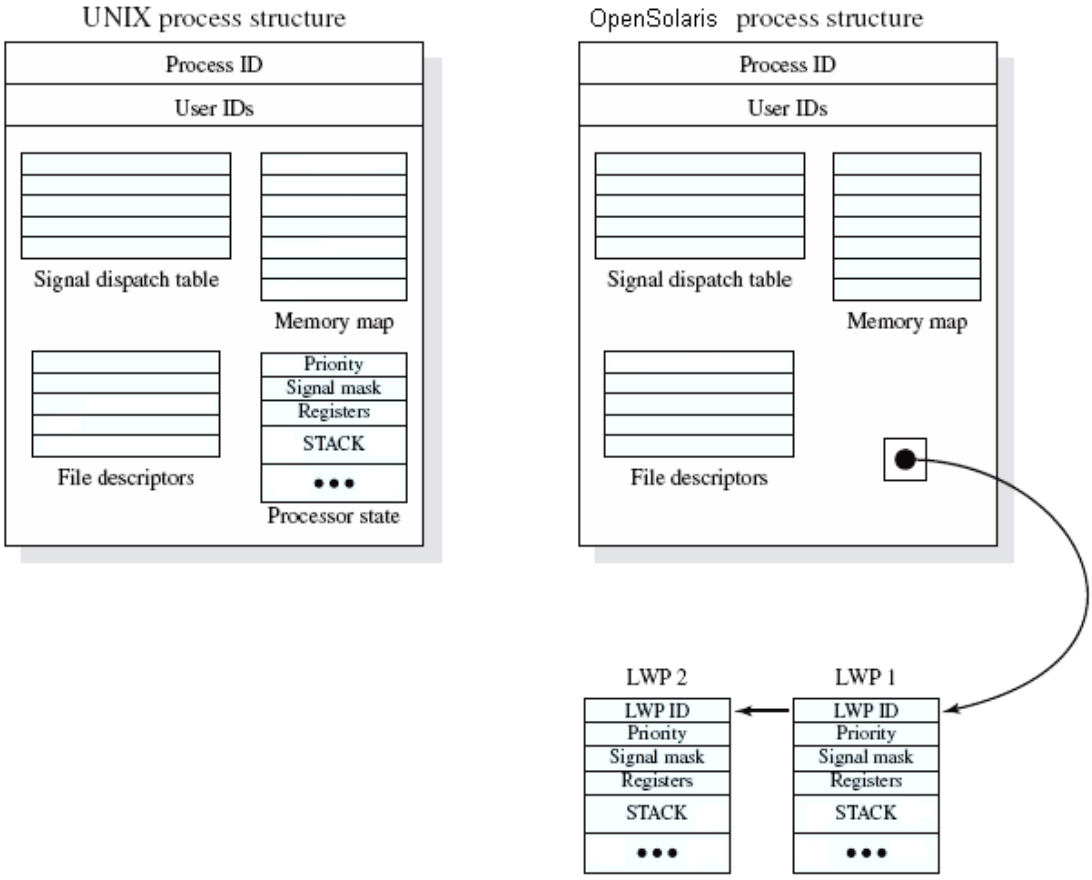

Figure 2 Comparison of Solaris Process structure and traditional Unix Process structure

Windows Vista

Vista process design is driven by the need to provide support for a variety of OS environments. Accordingly, the native process structures and services provided by the Windows Kernel are relatively simple and general purpose, allowing each OS subsystem to emulate a particular process structure and functionality. Here are some of the important characteristics of Windows processes:

- Windows processes are implemented as objects.
- An executable process may contain one or more threads.
- Both process and thread objects have built-in synchronization capabilities.

Figure 3[2] below illustrates the way in which a process relates to the resources it controls or uses. Each process is assigned a security access token, called the primary token of the process.

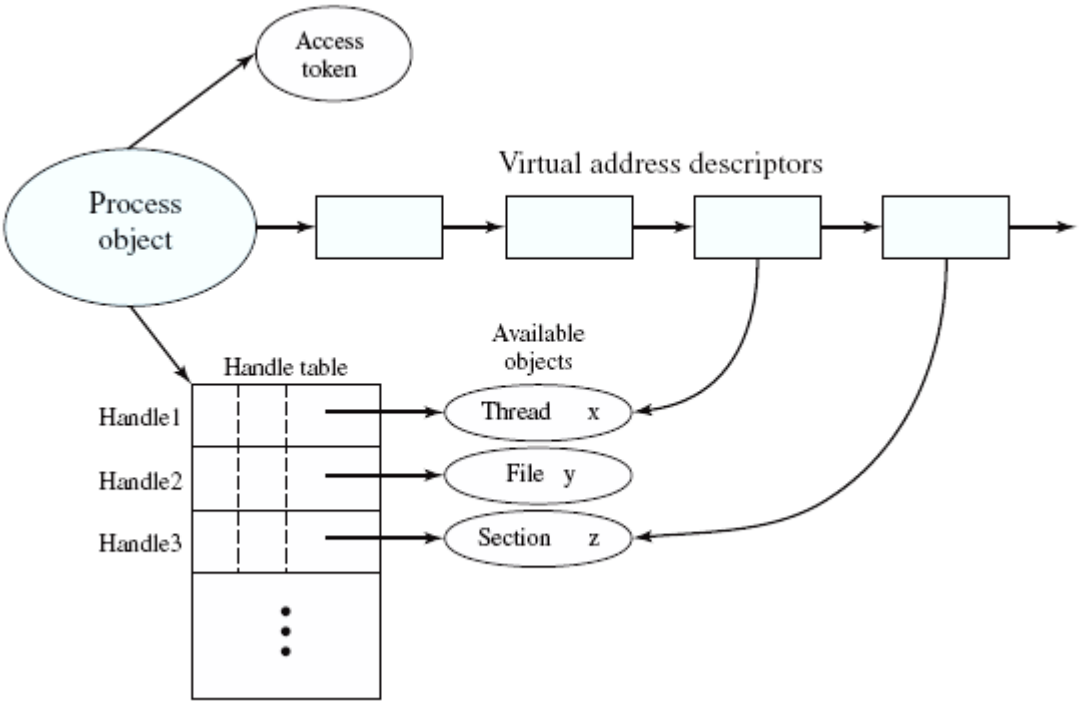

Figure 3 a windows process and its thread

When a user first logs on, Vista creates an access token that includes the security ID for the user. Every process that is created by or runs on behalf of this user has a copy of this access token. Windows uses the token to validate the user's ability to access secured objects or to perform restricted functions on the system and on secured objects. The access token controls whether the process can change its own attributes. In this case, the process does not have a handle opened to its access token. If the process attempts to open such a handle, the security system determines whether this is permitted and therefore whether the process may change its own attributes.

Also related to the process is a series of blocks that define the virtual address space currently assigned to this process. The process cannot directly modify these structures but must rely on the virtual memory manager, which provides a memory allocation service for the process.

Finally, the process includes an object table, with handles to other objects known to this process. One handle exists for each thread contained in this object.

In addition, the process has access to a file object and to a section object that defines a section of shared memory.

The object-oriented structure of Windows facilitates the development of a general-purpose process facility. Windows Vista makes use of two types of process-related objects: processes and threads. As OpenSolaris, a process is an entity corresponding to a user job or application that owns resources, such as memory, and

[2] Russinovich, M., and Solomon, D. Microsoft Windows Internals: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000. Redmond, WA: Microsoft Press, 2005.

opens files. A thread is a dispatchable unit of work that executes sequentially and is interruptible, so that the processor can turn to another thread.

Windows Vista supports concurrency among processes because threads in different processes may execute concurrently. Moreover, multiple threads within the same process may be allocated to separate processors and execute simultaneously. A multithreaded process achieves concurrency without the overhead of using multiple processes. Threads within the same process can exchange information through their common address space and have access to the shared resources of the process. Threads in different processes can exchange information through shared memory that has been set up between the two processes.
An object-oriented multithreaded process is an efficient means of implementing a server application. For example, one server process can service a number of clients.

Linux Kernel 2.6

A process, or task, in Linux is represented by a *task_struct* data structure. The *task_struct* data structure contains information in a number of categories:

1. Unlike Vista and opensolaris, processes in Linux are both containers and the schedulable entities; processes can share address space and system resources, making processes effectively usable as threads.
2. Also unlike Vista and OpenSolaris, Most services are implemented in the kernel, with the exception of many networking functions. Thus Linux kernel is relative bigger in size comparing former two OS.

Linux provides a unique solution in that it does not recognize a distinction between threads and processes. Using a mechanism similar to the lightweight processes of OpenSolaris, user-level threads are mapped into kernel-level processes. Multiple user-level threads that constitute a single user-level process are mapped into Linux kernel-level processes that share the same group ID. This enables these processes to share resources such as files and memory and to avoid the need for a context switch when the scheduler switches among processes in the same group.

Conclusion

Solaris and Windows both exist for dozens of years while Linux is very young and still has long way to go. Obviously, three OS are implemented following popular operating system theories. The most awkward obstacle is the ability for us to access the kernel implementation and debugging. Limited to my knowledge, I have no way to debug or trace thread and process in Windows Vista while OpenSolaris supplies abundant tools to observe kernel thread.
In the next paper, we will continue to discuss memory management and file system.