Synchronization Windows

- multithreaded kernal and supports real time application and multiprocessors.
- On uniprocess systems, It provides interrupt masks to protect access to global resources. It ~~process~~ protects access to global resource using spinlock. The kernal uses spinlocks only to protect short code segments like solaris.
- kernal ensures that while holding a spinlock, a thread will never be preempted.

Windows provide dispatcher object for thread synchronization according to several different mechanism including mutexes, Semaphores, events and timers. The system protects shared data by requiring a thread to gain owership of a mutex for accessing the data and when it is finished, releases thee Ownership.

Events: act as a conditional variable to notify a waiting thread when desired condition occurs

Timers: used to ~~modify~~ notify one or more thread when time expired

Dispatcher objects: may be either signaled state or a non-signale state.

Signalled state indicates that an objects is available and a thread will not block when acquiring the object.

Non-signalled state indicates that an objects is not available and a thread will block when trying to acquire the object

Synchronization in Linux

Process synchronization in Linux involves providing a time slice for each process so that they get the required time execution. The process can be created using the fork() command in linux. The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files and environment strings. However, they have distinct address spaces.

- Orphan Process.

Processes that still run even though their parent process has terminated or finished. Processes can be orphaned intentionally or unintentionally. Intentionally orphaned process runs in the background without any manual support. This is usually done to start an indefinitely running service or to complete a long running job without user attention.

An unintentionally orphaned process is created when its parent process crashed and terminated. Unintentional orphan process can be avoided using the process group mechanism

Daemon Process

Some processes run in the background and are not in the direct control of the user. They are known as daemon process. These processes are usually started when the system is bootstrapped and they terminate when the system is shut down. Usually the daemon process have a parent process that is the init process. The init process usually adopts the daemon process after the parent process forks the daemon process and terminates.

Synchronization in Solaris

Implements locks to support multitasking, multithreading & multiprocessing. It uses adaptive mutexes, conditional variable, semaphores, read-write locks, turnstiles to control access to critical section

Adaptive mutex: protects every critical data item which are only accessed by short code segments.

On a multiprocessor system it starts as a standard semaphore spin-lock. If the lock is held by a thread which is running on another CPU then the thread spins. If the lock is held by a thread which is currently in run state, the thread blocks going to sleep until it is awakened by signal of releasing the lock.

Solaris provides Read-Write lock to protect the data are frequently accessed by long section of code usualy in read-only manner. It uses turnsitiles to order the list of threads waiting to acquire either an adaptive mutex or read write lock. Turnstile is a queue structure containing threads blocked on a lock. They are per lock holding thread, not per objects. Turnstiles are organized according to priority inheritance which gives the running thread the highest of the priorities of the threads in its turnstiles to prevent priority inversion locking mechanisms are used by kernal is also used by user-level threads, so that the locks are available both inside and outside of the kernal. The difference is only that priority-inheritance is only used in kernal, user_level thread does not provide this functionality.