XXXIII Неделя науки СПбГПУ. Материалы межвузовской научно-технической конференции. Ч.IX: С.77-78, 2005.

УДК 681.3.07

He Xiao, Wu Bin (graduate students, Intelligent Systems Program),
M.V.Khloudova, PhD, associate prof.

POSIX THREADS AND SCHEDULING INTERFACE

Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system. In the UNIX environment a thread:
1. Exists within a process and uses the process resources;
2. Has its own independent flow of control as long as its parent process exists and the OS supports it;
3. May share the process resources with other threads that act equally independently (and dependently);
4. Dies if the parent process dies – or something similar.

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications.

In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations which adhere to this standard are referred to as POSIX threads, or Pthreads. Most hardware vendors now offer Pthreads in addition to their proprietary API's.

Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library – though the this library may be part of another library, such as libc.

There are several drafts of the POSIX threads standard. It is important to be aware of the draft number of a given implementation, because there are differences between drafts that can cause problems.

The POSIX 1003.1b scheduling routines, provided by **schedPxLib**, are shown in Table1. These routines let you use a portable interface to get and set task priority, get the scheduling policy, get the maximum and minimum priority for tasks, and if round-robin scheduling is in effect, get the length of a time slice. To understand how to use the routines in this alternative interface, be aware of the minor differences between the POSIX and Wind methods of scheduling.

Table 1: POSIX Scheduling Calls.

Call	**Description**
***sched_setparam*( )**	Set a task's priority.
***sched_getparam*( )**	Get the scheduling parameters for a specified task.
***sched_setscheduler*( )**	Set scheduling policy and parameters for a task.
***sched_yield*( )**	Relinquish the CPU.
***sched_getscheduler*( )**	Get the current scheduling policy.
***sched_get_priority_max*( )**	Get the maximum priority.
***sched_get_priority_min*( )**	Get the minimum priority.
***sched_rr_get_interval*( )**	If round-robin scheduling, get the time slice length.

POSIX and Wind scheduling routines differ in the following ways:

(a) POSIX scheduling is based on *processes*, while Wind scheduling is based on *tasks*. Tasks and processes differ in several ways. Most notably, tasks can address memory directly while processes cannot; and processes inherit only some specific attributes from their parent process, while tasks operate in exactly the same environment as the parent task. Tasks and processes are alike in that they can be scheduled independently.

(b) VxWorks documentation uses the term *preemptive priority* scheduling, while the POSIX standard uses the term *FIFO*. This difference is purely one of nomenclature: both describe the same priority-based policy.

(c) The POSIX scheduling algorithms are applied on a process-by-process basis. The Wind methodology, on the other hand, applies scheduling algorithms on a system-wide basis--either all tasks use a round-robin scheme, or all use a preemptive priority scheme.

(d) The POSIX priority numbering scheme is the inverse of the Wind scheme. In POSIX, the higher the number, the higher the priority; in the Wind scheme, the *lower* the number, the higher the priority, where 0 is the highest priority. Accordingly, the priority numbers used with the POSIX scheduling library (**schedPxLib**) do not match those used and reported by all other components of VxWorks. You can override this default by setting the global variable **posixPriorityNumbering** to FALSE. If you do this, the Wind numbering scheme (smaller number = higher priority) is used by **schedPxLib**, and its priority numbers match those used by the other components of VxWorks.