Paper summary

The user-level threads require no kernel intervention as managed by runtime libraries linked into each application's address space. It provides high performance and flexibility but lacks the functionality(I/O) that the operating system kernel thread can provide. But kernel threads provide poor flexibility. Scheduler activation provides the best of both the world. The scheduler activation incorporates a parallelism mechanism that includes the functionality of kernel-level threads with the performance and flexibility of user-level threads. The scheduler activation provides the necessary user-level context for running user-level threads, notifies the user-level of kernel events, and includes space for the kernel to save processor context. It has two execution stacks: kernel stack - used by the user-level when in kernel mode and the user stack - used by the user-level thread scheduler. The intercommunication takes place via upcalls. Thus, the Scheduler Activation serves as the mechanism for executing user-level threads and saving their context when preempted by the kernel. This enables the application the ability to schedule itself.

Strengths

1. Clear tables to help the user identify cases when kernel should call user space and vice versa.
2. Address the issue of kernel-level and user-level threads in a simple manner.
3. Useful for simultaneous access and manipulation of the same data.

Weaknesses

Handling of malicious/ buggy applications could have been in more detail.

Comments for author

Great work! My only question is, given the system's complexity, will different operating systems adopt this mechanism?
What if malicious process takes over and give fake resource usage?
Debugging section was unnecessary.