Peer review - Scheduler Activations: Kernel Support for the User-Level Management of Parallelism

María Fernanda Mora-Alba - MSc in Computer Science, ITAM

I. Summary

Threads play a key role in parallel computing. When it comes to high-performance and flexibility, user-level threads are better than kernel's. Aside from this fact, occasionally, kernel support is needed to correctly integrate user-level threads with other systems services. However, contemporary multiprocessor operating systems allow poor kernel support mainly because kernel threads provide the wrong abstraction of user-level thread management. The authors propose to combine the strengths of both approaches into a user-level thread package for performance, together with a kernel interface for functionality. The challenge is to allow communication between the kernel and the application. To achieve this the idea is to provide each application address space with a virtual multiprocessor, together with other information (number of processors and location of running threads). In this way the address space thread scheduler should perform thread scheduling and notify the kernel of the user-level events that can affect the allocation or use of processors. On the other hand kernel should allocate the processes and notify the thread scheduler of every event affecting the address space.

The paper also discusses the implementation, which is done through *scheduler activations*, defined as the execution context for vectoring control from kernel to the address space on a kernel event. The thread scheduler, in turn, takes this context to handle the event at user-level threads (modification of data structures and execution of threads and kernel petitions). Given that the kernel has no idea of the user-level data structures, *scheduler activations* are general enough to support any user-level concurrency model, not only threads. Finally the performance (speed and flexibility) is evaluated in the *Topaz* operating system. It was tested on forking a thread and running an application with a lot of interaction with the kernel.

II. Problem definition

Threads can be supported either at user-level or at the kernel but neither approach has been satisfactory. On the one hand user-level threads are managed by library functions linked to each application, requiring no kernel intervention, making them fast and customizable. The problem comes when integrating user-level threads -which are processors- with real OS activity (multiprogramming, I/O and page faults) which misrepresent the equivalence between virtual and real processors (scheduling ignores the state of the thread, thread blocking causes virtual processor to block) and can lead to low performance or errors. On the other hand kernel threads do not suffer from this misrepresentation because kernel itself assigns threads onto physical processors. The problem is that they are slow, hence not attractive for parallel computing. In an attempt to solve this, user-level threads were implemented on top of kernel threads as they are built on top of traditional processes, yielding the same problems as before. So the issue is to choose the customization, speed and restrictions of user-level threads, or the generality and slowness of kernel threads.

III. Contribution

The authors argue that user-level threads will always have a better performance because of the inherent costs of kernel threads: accessing thread management operations (crossing a level of abstraction is expensive) and generality (overhead on applications needing few features). So the task is to overcome the problems of level-user threads, which the authors claim is due to kernel's lack of support.

Scheduler activations are the proposed solution. They act as an instrument for running user-level threads in the way kernel's threads do. They also provide space in the kernel for saving the processor's context when a thread is stopped by the kernel. They allow communication between application and kernel as we explained before using an *activations system*. The user-level scheduler runs over the activations' user-level stack and calls to the kernel are done via the activations' kernel stack, making the communication indirect. Finally, the key advantage of an *activation* over a kernel thread is that when a thread is blocked a new scheduler activation is created asking the user-level system to remove the state of the thread and then telling the kernel that the activation can be reused.

IV. Critic

- Isolation from the kernel supports user-level speed but *activations* interact with the kernel each time a block occurs. In apps where the threads are constantly blocking kernel threads may be faster. Not adequate for I/O intensive apps.
- *Activations' upcall system* violates the protection layered structure by allowing calls from higher to lower levels.
- The point above allows the case of a malicious thread monopolizing the resources, starving the other processors.
- Overhead to maintain *scheduler activations* and increased system complexity due to the changes to kernel and user's code. How difficult is to implement it in different operating systems? The answer conditions it usability.

References

[1] Anderson, Thomas E., et al. ”Scheduler activations: Effective kernel support for the user-level management of parallelism.” ACM Transactions on Computer Systems (TOCS) 10.1 (1992): 53-79.