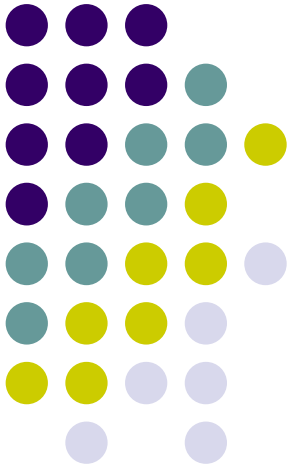
Scheduler Activations

THOMAS E. ANDERSON, BRIAN N. BERSHAD, EDWARD D. LAZOWSKA, and HENRY M. LEVY, "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism," ACM Transactions on Computer Systems, Vol. 10, No. 1, pp. 53-79, February 1992

http://cs.nyu.edu/rgrimm/teaching/sp07-os/activations.pdf

http://www.slideshare.net/kasunbg/scheduler-activations-effective-kernel-support-for-the-userlevel-management-of-parallelism



Issues

- User threads
  - A blocking system call blocks all user threads
  - A page fault blocks all user threads
  - Matching threads to CPUs in multiprocessors is hard
    - No knowledge about number of CPUs available to address space
    - No knowledge about when a thread blocks
- Kernel threads
  - Cost of performing thread operations
    - Create/exit/lock/signal/wait all require user↔kernel crossings
    - On Pentium III, getpid: 365 cycles vs. procedure all: 7 cycles
  - Cost of generality
    - Kernel threads must accommodate all "reasonable" needs
    - Kernel threads prevent application-specific optimizations
      - FIFO instead of priority scheduling for parallel applications

- Other issues
  - Just like processes, kernel threads do not notify user level
    - Block, resume, preempted without warning/control
  - Kernel threads are scheduled without regard for user thread state
    - Priority, critical sections, locks → danger of priority inversion
  - Matching kernel threads with CPUs
    - Neither kernel nor user knows number of runnable threads
  - Making sure that user-level threads make progress

Scheduler Activations

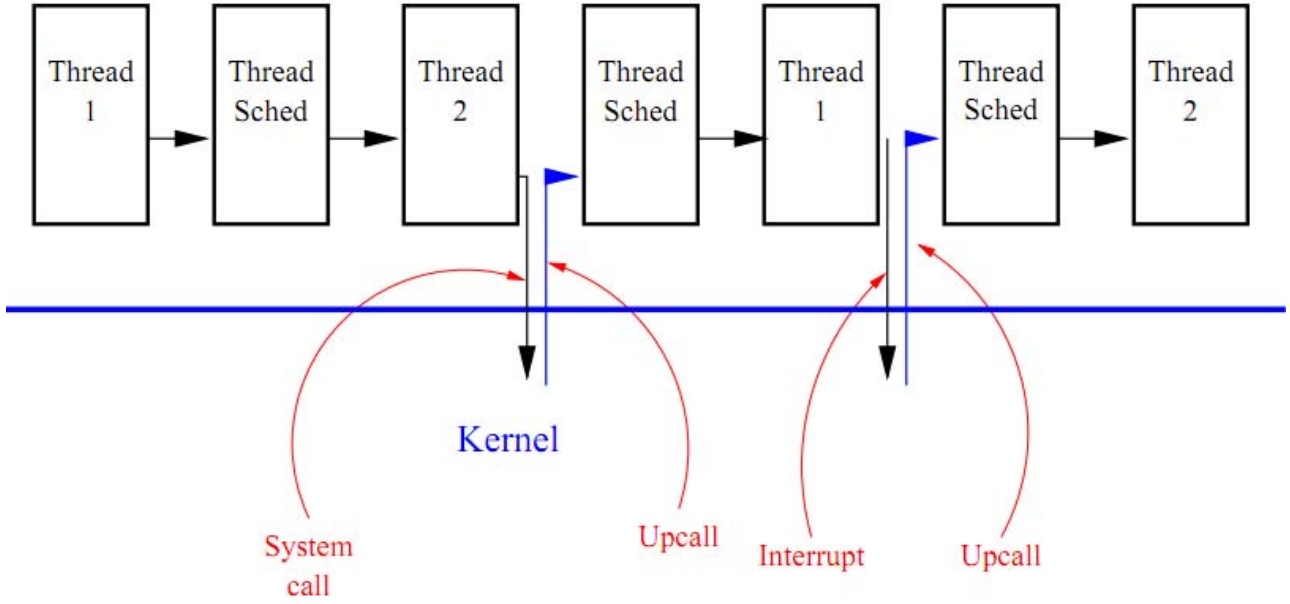
- Get the best of both worlds —
  - The efficiency and flexibility of user-level threads
  - The non-blocking ability of kernel threads (Poor integration with system services)
- Relies on upcalls
  - The term "scheduler activation" was selected because each vectored event causes the user-level thread system to reconsider its scheduling decision of which threads to run on which processors.
- What's an Upcall?
  - Normally, user programs call functions in the kernel or system calls
  - Sometimes, though, the kernel calls a user-level process to report an event
  - Sometimes considered unclean — violates usual layering

Scheduler Activations

- Upcalls
  - New processor available
  - Processor has been preempted
  - Thread has blocked
  - Thread has unblocked
- Downcalls
  - Need more CPUs
  - CPU is idle
  - Preempt a lower priority thread
  - Return unused activation(s)
  - After extracting user-level thread state

Scheduler Activations

- Scenario
  - Thread creation and scheduling is done at user level
  - When a system call from a thread blocks, the kernel does an upcall to the thread manager
  - The thread manager marks that thread as blocked, and starts running another thread
  - When a kernel interrupt occurs that's relevant to the thread, another upcall is done to unblock it

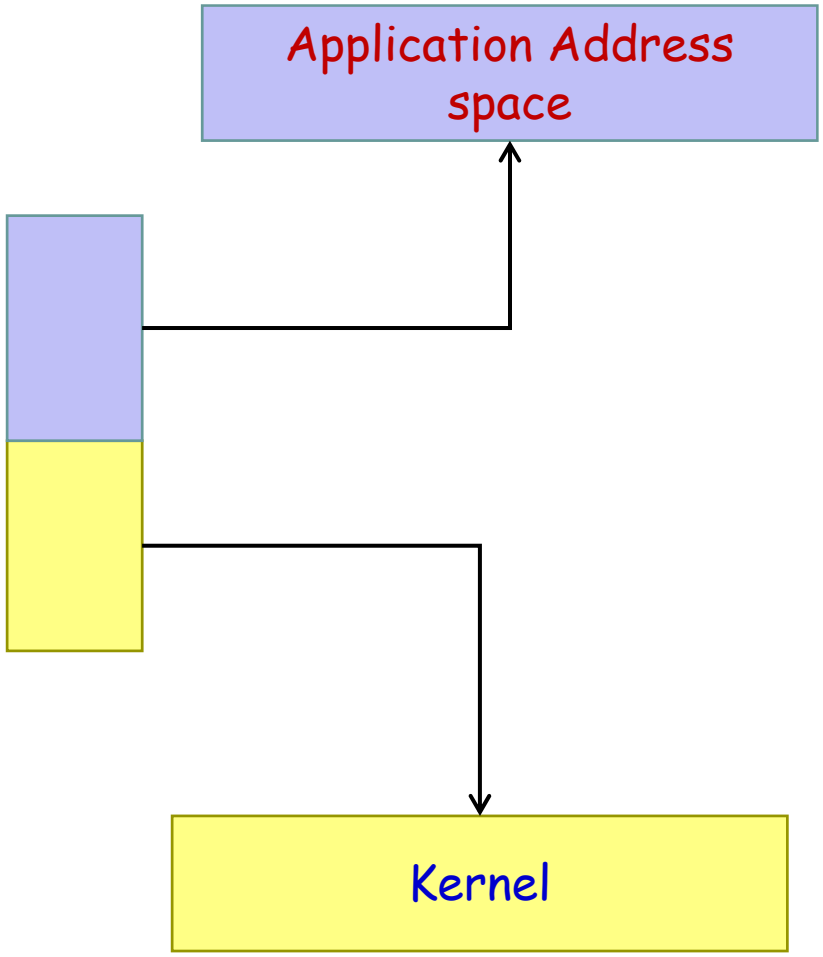


Scheduler Activations

- It serves as a vessel, or execution context, for running user-level threads, in exactly the same way that a kernel thread does.
- It notifies the user-level thread system of a kernel event.
- It provides space in the kernel for saving the processor context of the activation's current user-level thread, when the thread is stopped by the kernel (e.g., because the thread blocks in the kernel on I/O or the kernel preempts its processor).

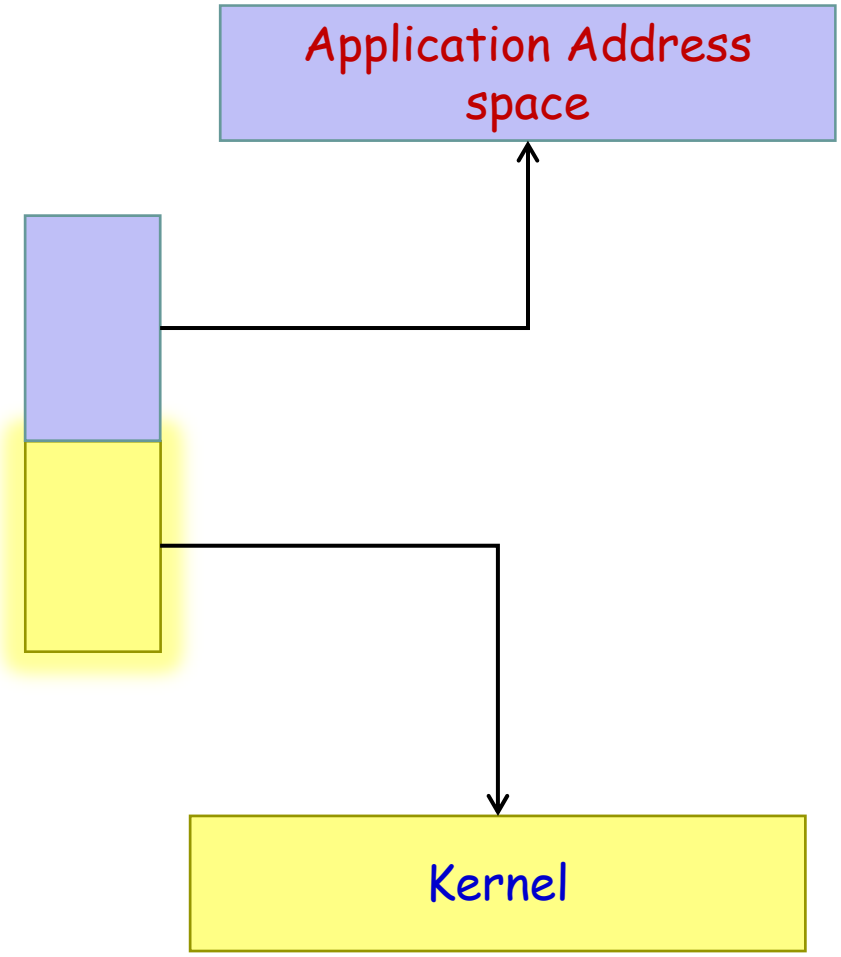
Scheduler Activations

- Two execution stacks
- One mapped into the kernel
- The other mapped into the application address space.



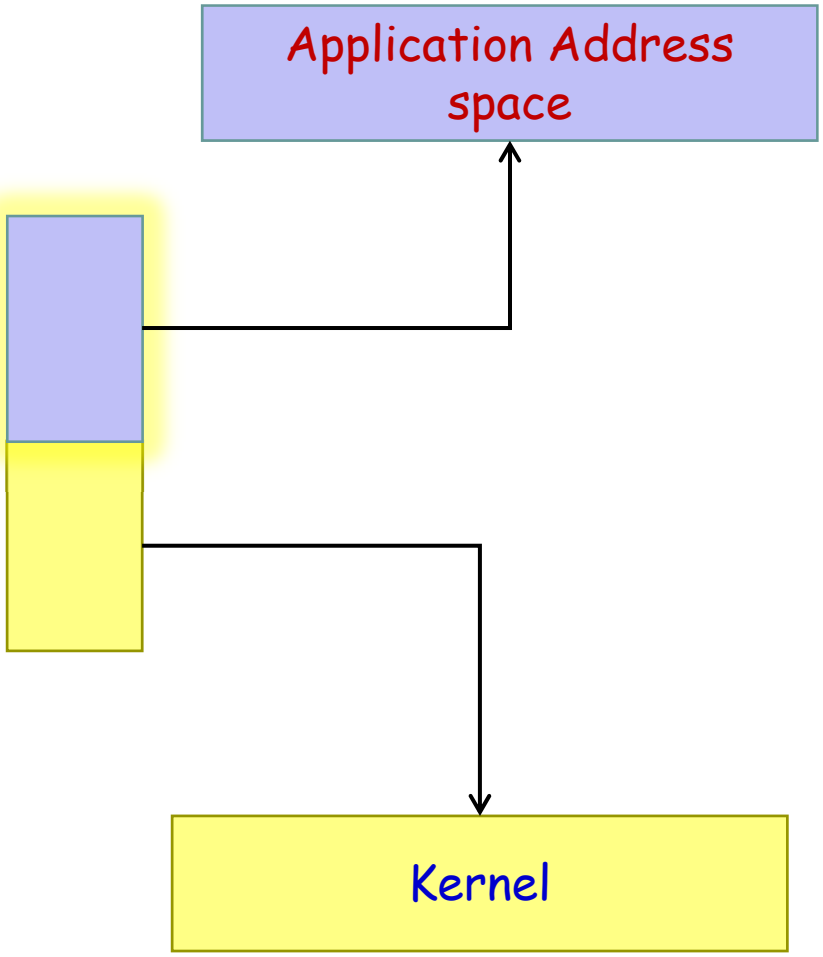
Scheduler Activations

- Kernel stack is used whenever the user thread running in the scheduler activation's context executes in the kernel (e.g. system call)
- The kernel also maintains a control block for each activation (akin to a thread control block) to record the state of the scheduler activation when its thread blocks in the kernel or is preempted.

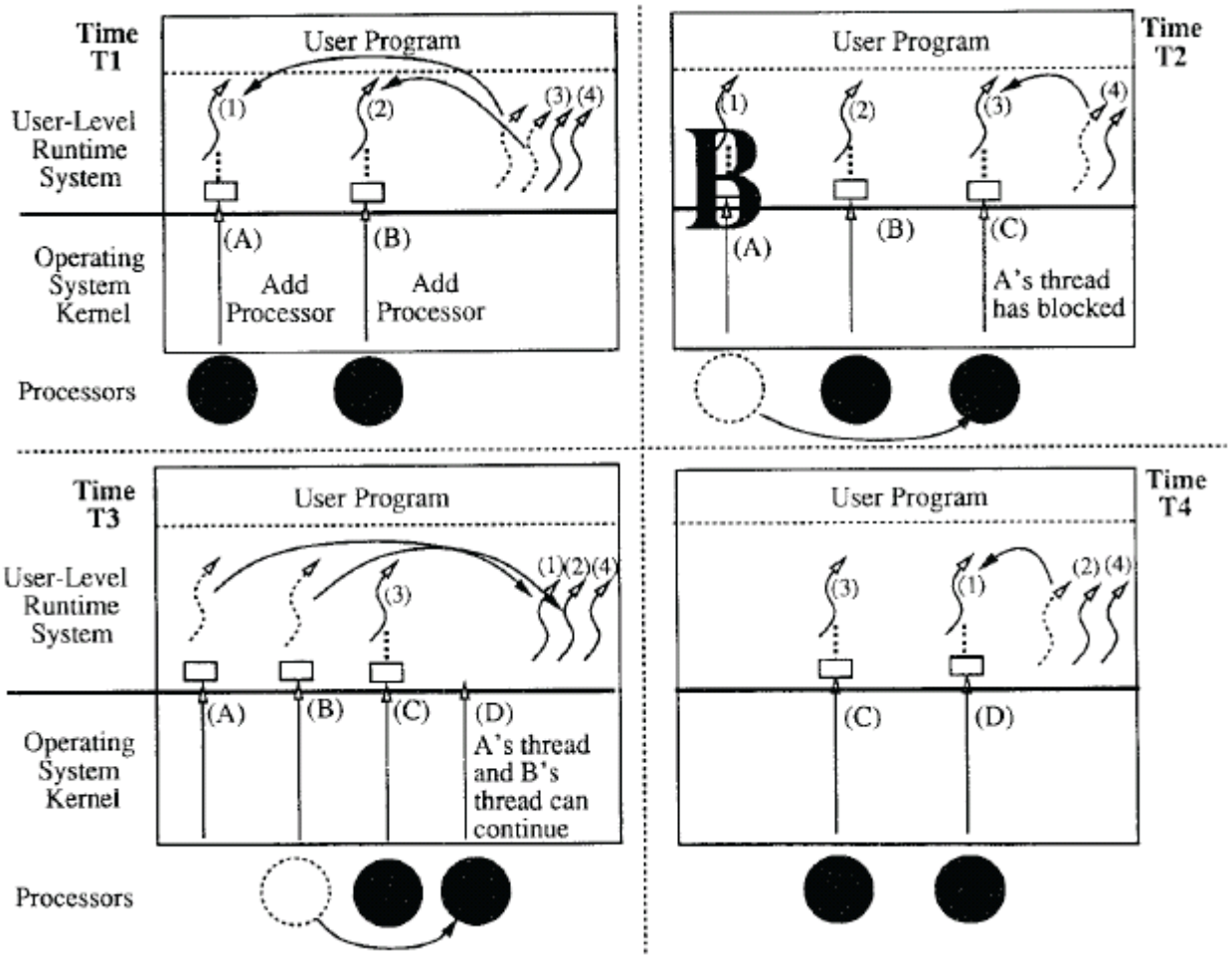


Scheduler Activations

- The user-level thread scheduler runs on the activation's user-level stack and maintains a record of which user thread is running in which scheduler activation.
- Each user thread is allocated its own stack when it starts running



An Example



Straight arrows represent scheduler activations, s-shaped arrows represent user-level threads, and the cluster of user-level threads to the right of each pane represents the ready list.

At Tl, the kernel allocates the application two processors. On each processor, the kernel upcalls to user-level code that removes a thread from the ready list and starts running it.

At time T2, one of the user-level threads (thread 1) blocks in the kernel. To notify the user level of this event, the kernel takes the processor that had been running thread 1 and performs an upcall in the context of a fresh scheduler activation. The user-level thread scheduler can then use the processor to take another thread off the ready list and start running it.

At T3, the 1/0 completes. Again, the kernel must notify the user-level thread system of the event, but this notification requires a processor. The kernel preempts one of the processors running in the address space and uses it to do the upcall. (If there are no processors assigned to the address space when the 1/0 completes, the upcall must wait until the kernel allocates one). This upcall notifies the user level of two things: the 1/0 completion and the preemption. The upcall invokes code in the user-level thread system that (1) puts the thread that had been blocked on the ready list and (2) puts the thread that was preempted on the ready list. At this point scheduler activations A and B can be discarded.

Finally, at T4, the upcall takes a thread off the ready list and starts running it.