# CSE 60641: Operating Systems

- **Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Hank M. Levy, SOSP '91**
  - **What is this paper trying to achieve?**
    - **User level threads**
    - **Kernel level threads**
    - **Scheduler activation**

## User level threads

- Excellent performance (no crossing user/kernel boundary)
  - No protection issues because all threads belong to the same process
- Cost of generality: user threads can be tuned to a particular application
- Problems:
  - blocking call, kernel thread allocated to the user thread cannot be reclaimed
    - Allocate more virtual processors (unfair)
  - Kernel might interfere with critical sections
  - Processors cannot be returned

## Solution: Scheduler activation

- A mechanism for kernel and user level to cooperate
  - Kernel makes an upcall with a scheduler activation
  - You can either keep the activation, or perform the task informed by the activation.
  - User level -> kernel is still system call

- Number of scheduler activations = number of virtual processors assigned to a process
  - Application is free to implement any scheduling policy using the assigned activations
  - Kernel will notify user if any thread blocks

## Kernel upcalls because of

- Add this processor
- Processor has been preempted
  - Return to the ready list of the user-level thread that was executing in the context of the preempted activation
- Scheduler activation has blocked
  - Blocked scheduler is no longer using its processor
- Scheduler activation has unblocked
  - Ready to the ready list the user level thread that was executing in the context of the blocked activation.
  - New activation includes processor context the newly unblocked one and the one that was preempted

## From user space to kernel

- Add more processors
- Processor is idle
- Together, they can allow the kernel to give and take away processors dynamically

- User level priority scheduling:
  - Deschedule a lower priorty thread
  - If the lower priority thread is in critical section, don’t do it
    - Detected using application maintained flags

- Performance good:
  - Upcall latency: 5x kernel threads

## Lessons learnt

- Can kernel/user partnership make sense for other things? If so, how general can they be?