Outline

- Project status

- SOSP 1991
  - Using Continuations to Implement Thread Management and Communication in Operating Systems *Richard P. Draves, Brian N. Bershad, Richard F. Rashid, Randall W. Dean*
  - 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*

Continuations

- Richard Draves – Leads Systems and Networking group at Microsoft Research
  - Mach, Rialto
- Brian Bershad – Univ. of Washington CMU UW
  - Perennial SOSP contributor….
- Richard F. Rashid – CMU Head, MS Research
  - Mach, ….
- Randall Dean – CMU, VP, Mercury Systems?
  - Mach, ?

Processing within kernel

- Process model
  - Multiple threads within kernel
  - Unique kernel stack - rescheduled and descheduled transparently
  - E.g. UNIX
  - Easy to use as blocking is transparent
  - Cannot optimize unwanted “stack” space
- Interrupt model
  - Single per-processor stack
  - Threads explicitly save state before blocking
  - E.g. Quicksilver, V

Possible solution

- User level thread - one kernel thread for many user level threads
  - At least need one kernel level thread
  - Blocked kernel threads still wasted resources

- Continuations
  - Provide code to save state
  - Behaves like process model
  - Performance of interrupt model
  - Allows further optimizations

Key idea

- Optimizing Mach 3.0
  - Application level representation of state while blocked
  - Application code to restore stack
  - Optimizations to reduce continuation operations
    - Tradeoff stack space for complexity (application code)
- Is this relevant?
  - Current processors have lots of memory, so why bother?
  - Per processor kernel stack
    - reduce cache and TLB misses
- Software engineering concerns?
- Interrupt driven, co-routine style services
  - Much current work in MS Research and other places

Scheduler activations

- Tom Anderson
  - Developed Nachos instructional OS
  - UW → Berkeley → UW
- Brian Bershad
- Ed Lazowska
  - UW
- Hank Levy
  - UW - 17 Ph.D, ACM/IEEE Fellow etc. etc.

User level, Kernel level threads?

- User level threads
  - Fast - the kernel does not need to know when there is a switch
  - Flexible - each process can use its own scheduler
  - Can block - Kernel does not know about the existence of user level threads
- Kernel level threads
  - Can block - kernel can schedule other kernel level threads
  - Slower - protection boundry crossed

Scheduler activation

- Cooperative mechanism
  - Kernel informs the user process of number of virtual "processors" as well as change in the number of processors
  - User threads use these processors without informing the kernel on the scheduling decisions
  - Scheduling decision by the user level library + Processor allocation, blocked thread processing etc by kernel scheduler activation mechanism
    - User thread can request and relinquish virtual processors
    - Upcalls from kernel to application
    - System calls from application to kernel