KSE and Threads

Some Background

- Thread: stream of execution
- Process: container for a thread
  - Address space (memory map)
  - File descriptors
  - Credential
  - ...
- Traditional Unix process is 1:1 mapping
- Multithreaded: n:1

User-level Threads

- All thread management in libraries outside kernel
- Kernel is unchanged, ignorant (sees only processes)
- Threads voluntarily pass processor among themselves (lightweight, +)
- Programmer directly in control of parallelism (+)
- Block one thread, block all (syscalls, -)
- Cooperating threads get only 1 processor (-)
Kernel-level Threads

- Kernel knows all about threads
- Kernel, not user, controls thread scheduling (-)
- Heavier-weight switch between threads (context switch, -)
- Syscalls need not be wrapped (+)
- True parallelism possible on an SMP machine (+)

How to get best of both?

- Scheduler Activations (Anderson, et al., 1992)
- SA are "almost" like kernel threads
  - Understands blocking
  - Gets benefits of kernel threads
- ...yet the user is still in charge
  - Scheduling
  - Switching between threads in the same process is fast
- FreeBSD Kernel Scheduling Entities (KSE) are Scheduler Activations

The Basics of KSE

- A KSE is a virtual processor: it can run a thread
  - It can also act a bit on its own, as we’ll see
- A KSE is assigned to a process by the kernel
  - Gives a process the ability to run a thread
  - Doesn’t decide which thread to run
- The process decides which of its threads gets the KSE
Mapping User Threads to KSE to Processors

KSE Basics II

- Processes start out with on KSE for a single thread (classic unthreaded unix process)
- Processes can request additional KSE from the kernel
  - But not more than there are physical CPUs in the system

But Wait!

- So far, KSE sound just like classic Kernel Threads
- The difference is in the upcall mechanism
- Whenever a kernel event takes place, the kernel calls up into the user-level thread scheduler to decide what to do
What’s a Kernel Event?

- A blocking system call
- A page fault
- A higher-priority thread is ready to return to user space, and the current thread is returning
- A signal being delivered to the process

What Happens on a Kernel Event?

- KSE becomes unassigned
- An upcall is done to the user library to make a scheduling decision
- Upcalls require the use of mailboxes

KSE Mailboxes (see <sys/kse.h>)

- km_func field points to upcall function
- km_curthread points to the thread mailbox for the currently executing thread
  - Manipulated by both kernel and user routines
  - If NULL, no upcalls performed
    - Standard (old-fashioned) Unix process
    - Kernel may set to null before calling km_func; user-level scheduler sets to new value
Thread Mailboxes

- \texttt{tm\_context} stores thread context when blocked in user space
- Flags field allows setting \texttt{TMF\_NOUPCALL} which prohibits upcalls during critical sections

KSE Groups

- The smallest granularity that can have a priority assigned by the kernel
- Share accounting information

For More Information

- See the KSE man page
- \texttt{kse\_create()}
- \texttt{kse\_exit()}
- \texttt{kse\_release()}
- \texttt{kse\_switching()}
- \texttt{kse\_thr\_interrupt()}
- \texttt{kse\_wakeup()}

---

---

---

---

---

---

---

---

---