User Level Interprocess Communication for Shared Memory Multiprocessors
Motivation

We want to build modular systems
We want each module to be in its own protection domain
Its expensive to communicate across protection domain boundaries
- system call cost vs procedure call cost
LRPC

Didn’t LRPC just solve this problem?
Its faster than RPC, but still much slower than same address space procedure call
Why?
LRPC Involves the Kernel

Switching the CPU from one address space to another requires privileged instructions
- LRPC asks the kernel to do that work

Switching threads may or may not involve the kernel
- LRPC uses kernel threads
URPC

If we have more than one CPU, we can leave one CPU in the caller address space and the other in the callee address space.
- switching is no longer necessary
- kernel intervention is not necessary

If we have user level threads, no kernel intervention is needed to switch threads.
How Does It Work

How can one user thread in one address space communicate with another in a different address space?

Solution: map a shared memory region into both address spaces
- caller writes to that region
- callee reads from it

The kernel is needed to create the mapping, but not to use it!
How Are Calls Received?

How does the callee thread know when a caller has written to the shared region?

Solution: it polls

Polling frequency influences call delay!
How Are Replies Received?

How does the calling thread know the callee has written the result to the shared region?

Solution: it polls

Polling frequency influences call delay!
CPU Utilization

How many CPUs are we using to execute one control flow?

Is this cheating?

If we switch to a different user level thread once a call has been issued, how does this affect our poling frequency?
Conclusion

URPC is a communication mechanism that trades delay for throughput.

We now have scheduling, synchronization, and cross-address space communication implemented at user level.

What is left in the kernel?

Do we need a kernel?