Integrated Task and Stack Management
Questions

What do these terms mean?

- process, thread, kernel-level thread, user-level thread, fiber?
- user-level thread stack, user stack, kernel stack?
- Address space?
Questions

- In what way are user-level threads “faster” than kernel-level threads?
  - Aren’t we just layering abstractions on abstractions?

- Why can’t you do preemptive user-level thread scheduling without special kernel support?
  - Does this mean you can only do sequential task management?
  - How would you support cooperative task management?
Questions

- Is synchronization needed for cooperative user-level scheduling on uni-processors?
  - What precautions must you take?
  - Is synchronization needed on multiprocessors?

- How are synchronization operations such as spin-locks and semaphores implemented, i.e., how do you spin and how do you block?
  - For user level threads?
  - For kernel level threads?
Questions

- How might a user-level thread library support synchronous I/O calls for its threads without losing control of the CPU due to blocking?
  - How does this approach behave during page-faults?
  - Can it deal with preemptions?

- Do user-level thread libraries do “stack ripping”?
Questions

- What stack management model do most OS kernels export?
  - How are I/O and page faults events handled?

- What differences are there between virtual CPUs (threads) and real CPUs (hardware)?
  - How are I/O and page fault events handled?
  - How is time managed?
Questions

- What synchronization-related issues do you have to worry about when writing an interrupt handler?
  - Do these occur with scheduler activation handlers?

- Why must the user-level thread library be reentrant?
  - How is its pattern of entry points and synchronization concerns similar to or different from kernel-level scheduler code?
    - With scheduler activations and without scheduler activations
Questions

- Why does the kernel delay delivery of a second page-fault SA until the first page-fault SA on the same address has completed?

- Why might a user level thread scheduler choose to delay context switching a user level thread that is holding a spin lock?
  - How can it do this in the presence of kernel preemption and notification via scheduler activations?
Questions

- In what ways do scheduler activations mix manual and automatic stack management models?
RPC Questions

- What programming model / code structure does RPC enable?
  - What is the underlying model / structure?

- What are RPC stubs and what do they do?
  - How do they relate to adaptors discussed previously?
  - Is there any stack ripping involved?

- Can RPC pass parameters by reference?
General Questions

- What alternative approaches could they have taken to implement a shared memory programming abstraction over a distributed infrastructure?
  - And what are the relative advantages and disadvantages compared to RPC?