

CS 518

Scheduler Activations: User/Kernel Interactions

Randy Wang
Fall 2002
Princeton University

Philosophy

- Exokernel: an interface issue
 - User maintains flexibility with the resource it has
 - Kernel controls multiplexing resources among different users
 - What is the right interface for doing this?
- Scheduler activation: a specific case study of the general Exokernel philosophy

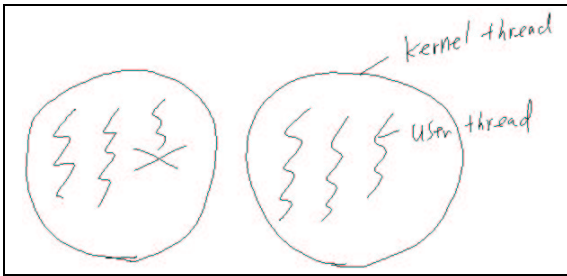
Limitations on Concurrency

- Hard to design and debug
- Application characteristics
- Latency of communications

Existing Approaches

- Kernel threads
 - too expensive to create
 - context switches slow
 - lacks user-level flexibility

Existing Approaches (cont.)



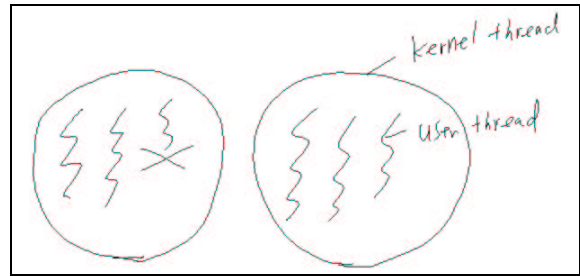
- User threads on kernel threads
 - like to have one kernel thread per cpu
 - light-weight scheduling/synchronization at user level
 - bad interactions with kernel
 - block I/Os, page faults
 - block: app may lose cpu if not enough kernel threads
 - unblock: app may lose scheduling decision if too many kernel threads
 - multi-programming
 - app may lose scheduling decision

CS518

4

Randy Wang

Scheduler Activation Principles



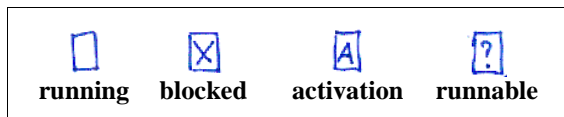
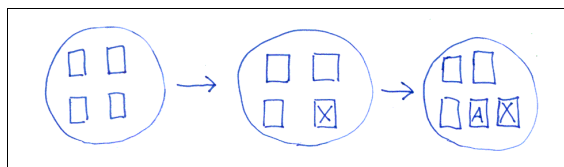
- Real problem: two schedulers
 - User scheduler: schedules based on application semantics
 - Kernel scheduler: schedules based on kernel events and multi-programming
- Fundamentally, a mechanism to pass info back and forth between the user and the kernel
 - Kernel tells how many processors are available
 - User given chance to decide what to do with them
 - User tells how many processors it needs
- ~~Fixed number of threads~~
 - > Fixed number of running threads

CS518

5

Randy Wang

(1) Blocking



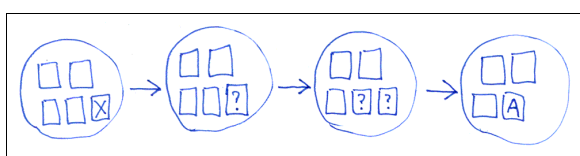
- Kernel gives new activation
- App decides what to do with it
- Maintains level of concurrency

CS518

6

Randy Wang

(2) Unblocking



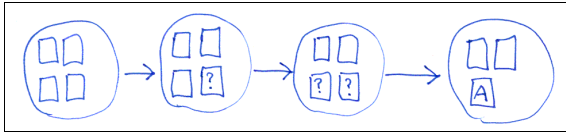
- Kernel preempts a running thread: two runnable threads now
- Kernel gives register state in new activation
- App uses it to decide what to do
- Maintains level of concurrency
- What exactly is an “activation”?
 - Basically saves thread state in the kernel
 - Exists from blocking to waking-up
 - Activation: kernel notifies user
 - Needs a processor to run on when “activated”

CS518

7

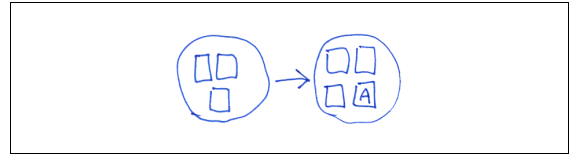
Randy Wang

(3) Preemption



- Just like previous picture (2), without an unblocked thread though
- App's concurrency goes down by 1
- What happens on boundary cases?

(4) Resumption



- Just like blocking case (1), without a blocked thread though
- App's concurrency goes up by 1
- What happens on boundary cases?

Loose Ends

- The above: kernel tells app. Also need: app tells kernel its desired level of concurrency
- Cost of activations
 - Relatively high compared to user-level events
 - Relatively low compared to kernel context switches
 - Doesn't happen that often: happens only as a result of "other" expensive ops

Summary

- Goals:
 - User threads: common case
 - fast performance
 - retains flexibility
 - Kernel threads: rare case
 - gets kernel events right
 - Want to have the best of both worlds
- Mechanism:
 - Exchange events between kernel and user that can/should influence scheduling decisions