

Outline

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel

Simon Kågström, Lars Lundberg, Håkan Grahn
{ska,llu,hgr}@bth.se

Blekinge Institute of Technology
Sweden

- Background
- Traditional approaches
- The application kernel approach
- Prerequisites
- Implementation
- Example of the operation
- Performance
- Conclusions

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 116

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 216

Background

- We are working together with a manufacturer of industrial systems in Sweden
- The company has a mature uniprocessor (IA-32) kernel, and is currently investigating a port to SMP hardware
- The port has proven to be hard
 - Code size
 - OS kernels are intricate
 - Hard to get correct

We would like to implement the SMP support without modifications to the OS kernel

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 316

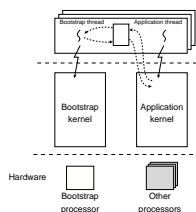
Traditional parallelization approaches

- Monolithic kernels
 - The kernel contains a large part of the operating system
 - Many parts of the kernel needs to be protected
- Microkernel-based systems
 - Microkernels allows better isolation of OS parts
 - A large part still needs to be changed
- Alternative approaches
 - AsyMOS [Muir98], application processors and device processors
 - Master-slave systems, one processor is the “master” processor (OS operations), the rest are “slaves”

All require extensive modification of the kernel

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 416

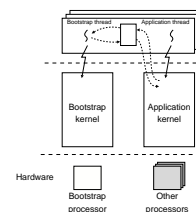
The application kernel approach (1/2)



- Two kernels, similar to master-slave systems
- A small kernel (*application kernel*, *apkern*) runs on all processors except the bootstrap processor
- Kernel code still only runs on the bootstrap processor

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 516

The application kernel approach (2/2)



- Applications run on top of the custom kernel on the *application processors*
- For system calls, applications communicate with the bootstrap processor

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel – p. 616

Benefits

- *apkern* is small
 - Simple to construct and validate
 - Generic solution, easily portable
- The original kernel is kept largely unchanged
 - We don't need to touch complex kernel code (except for certain page table updates)
 - Changes to the original kernel automatically propagate to the new system

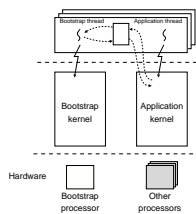
A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 716

Prerequisites

- Access to the processor supervisor mode
 - Needed in order to startup secondary processors etc.
- The possibility of injecting code into a running kernel is needed (drivers, modules etc.)
- Knowledge of some OS behavior:
 - How system calls are handled (parameter passing etc.)
 - How faults etc. are handled. For instance, is a process killed on a divide-by-zero?
 - How a process is started (this needs to be modified)

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 816

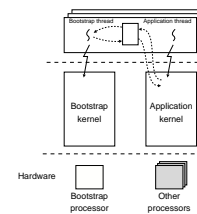
Implementation (1/3)



- We implemented the approach for a small kernel
- Every process has two threads
 - The application thread running on the *apkern*
 - A bootstrap thread running on *bootstrap kernel*

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 916

Implementation (2/3)



- A shared memory area is used for communication
 - When *apkern* gets a syscall/exception, it enters information about it in the shared area
 - When the bootstrap thread awakes next time, it checks the shared area and issues the corresponding event

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 1016

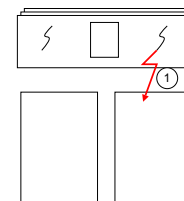
Implementation (3/3)

- The bootstrap thread runs a loop:


```
while(1) {
    if (message msg in the shared area)
        handle_message(msg);
        apkern_activate_thread();
    else
        reschedule();
}
```
- The syscalls in the application remain unchanged, *apkern* contains an implementation of them
- *Apkern* is implemented as a device driver or similar entity running in supervisor mode

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 1116

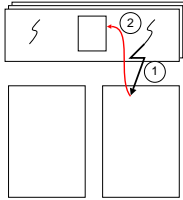
Example, system call



1. The application thread issues the system call
2. *apkern* enters information about the system call into the shared area
3. The bootstrap thread finds the entry in the area
4. The bootstrap thread issues the corresponding call

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel -- p. 1216

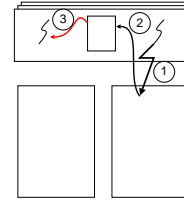
Example, system call



1. The application threads issues the system call
2. *apkern* enters information about the system call into the shared area
3. The bootstrap thread finds the entry in the area
4. The bootstrap thread issues the corresponding call

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 12/16

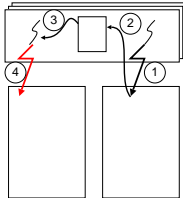
Example, system call



1. The application threads issues the system call
2. *apkern* enters information about the system call into the shared area
3. The bootstrap thread finds the entry in the area
4. The bootstrap thread issues the corresponding call

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 12/16

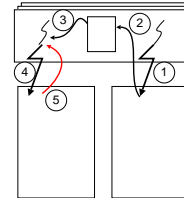
Example, system call



1. The application threads issues the system call
2. *apkern* enters information about the system call into the shared area
3. The bootstrap thread finds the entry in the area
4. The bootstrap thread issues the corresponding call

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 12/16

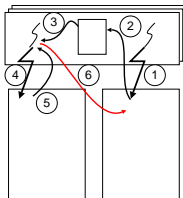
Example, system call



5. The bootstrap kernel handles the system call and returns to the bootstrap thread
6. The bootstrap thread tells *apkern* that the call has been handled
7. *apkern* returns control to the application thread

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 13/16

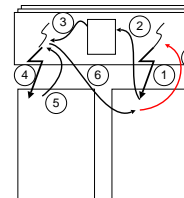
Example, system call



5. The bootstrap kernel handles the system call and returns to the bootstrap thread
6. The bootstrap thread tells *apkern* that the call has been handled
7. *apkern* returns control to the application thread

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 13/16

Example, system call



5. The bootstrap kernel handles the system call and returns to the bootstrap thread
6. The bootstrap thread tells *apkern* that the call has been handled
7. *apkern* returns control to the application thread

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 13/16

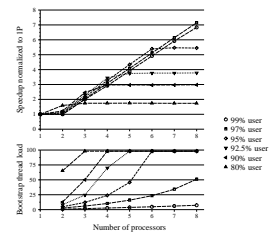
Performance, 1/2

- We evaluated the performance using micro benchmarks running on the Simics full-system simulator
- Latency measurements: about 1200 instructions/syscall added by our approach
- The latency mainly depends on the bootstrap kernel
- Also measured throughput of system calls with different lengths

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 14/16

Performance, 2/2

- Throughput measurements (around 4500 instructions per system call)



- Upper graph: the speedup vs uniprocessor operation
- Lower graph: the load on the bootstrap processor

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 15/16

Conclusions

- We have designed and implemented an alternative SMP porting method for operating systems
- SMP support for systems where the kernel is hard to modify, lower implementation complexity
- Uniprocessor kernel updates are automatically propagated to the SMP system
- Scalability is OK with long system calls and dominant user mode execution
- We plan to test the idea on a full-scale operating system (probably Linux)

A novel method for adding multiprocessor support to a large and complex uniprocessor kernel - p. 16/16