summary

Multiplexing User Address Space on x86 and x86_64 compatible processor family

Coly Li
coyli@novell.com

Content

- Transparently Multiplexing User Address Space
- New Issues On New Conditions
- Improvement Based On Liedtke's Work
- Methods Comparison
- Key Points Of Implementations
- Benchmark And Results Evaluation
- Difficulties To Be Overcome

Transparently Multiplexing User Address Space

Basic idea is from Liedtke's creative work which uses small address space to emulate TLB tagging.

- Small address spaces are shared by all large address space processes, in range of [2.5GB, 3GB).
- Large address space process A switches to small address space process S (in order to perform IPCs), no TLB flushing in context switching needed.
- Small address space process S switches to small address space process M, no TLB flushing in context switching needed.
- Only when small address space process S switches to a large address space process B, which is not the lastest large address space process (e.g. A) switched to small address space processes, the TLB flushing is needed.
- TLB flushing is needed for context switching between large address space processes.

Transparently Multiplexing User Address Space (Cont.)

- This scheme achieves perfect IPC performance by avoiding unnecessary TLB flushing in context switching for passing messages.
- Requirements for this optimization scheme:
  - Non-tagging TLBs
  - Address space protection based on segments
  - Memory management based on page mapping
  - Frequently IPCs from large address space processes to small address space process, or from small address space processes to small address space processes.
- On Intel Pentium processor family or compatible platforms, all of these requirements are met.

Transparently Multiplexing User Address Space (Cont.)

- On L4 implementation on x86 platform, e.g. L4Ka::Pistachi, system calls are served by L4Linux, not native L4 server processes.
- Most of the messages for system calls can only be passed between user space applications and L4Linux server.
- L4Linux can be placed in shared small address space.
- When
  - large address space A passes message to L4Linux.
  - L4Linux passes messages to large address space A which is the last large address space process sent message to L4Linux.

  IPC performance can be improve by avoiding TLB flushing in context switching.
- L4Linux running environment implementation is simple and efficient.

New Issues On New Conditions

New platforms, new applications, new conditions

- Multicore processor, hundred cores in one socket.
- Virtualization, dozens of virtualized OS instances running.
- Popular huge address space, 64bit x86 on Desktop/Laptop.

Decade ago, these platforms and applications existed already. But today, they become more and more popular, even appear on Personal Computer and embedded hardwares. These are new conditions to microkernel design.

Can Liedtke's idea still work well under the new conditions ?
Can we KEEP improving IPC performance on MICROKERNEL based system ?

New Issues On New Conditions (Cont.)

Because 2nd general microkernel is hardware concreted, these questions are only be reviewed on x86 and x86_64 platforms. Other hardwares will not be concerned in this document.

Issues on x86 and x86_64

- Liedtke's small address space idea is based on context switching with segments. On x86_64 , there is no base and limit for segments, segments are only used for privilege restriction, can not provide protection for separated small address space.
- On x86_64 there are only user/supervisor privilege levels by page based protection. When complexing address space with large address space processes,
  - If L4Linux running in user privilege, there is possibility for user process to destroy L4Linux server.
  - If L4Linux running in supervisor privilege, there is possibility for L4Linux to destroy ukernel.

New Issues On New Conditions (Cont.)

- In order to provide low latency for interrupt handling or system call, implement the handlers as threads is a good choice. On multicore CPUs, assigning each core with a handler thread can gain better performance. In L4Linux scheme, this will cause too much threads to be generated in a single server. Huge number of threads in once process has less flexibility for scheduler.
- In conditions that running dozens of virtualized OS on multicore processors, there are different priorities, different realtime deadline, different resource quota, more frequent system calls. A single L4Linux server with huge number of threads can not meet the requirements.

Liedtke's transparently user address space multiplexing scheme can not work on x86_64 any more. The L4Linux can not provide satisfied performance running environment on multicore processors (plus virtualizations), too.

Improvement Based On Liedtke's Work

Inherit Liedtke's emulating tagged TLB idea, reference Mach's co-located server conception.

Divide processes of L4 into 4 categories:
1, Critical, trusted.
2, Critical, untrusted.
3, Non critical, trusted.
4, Non critical, untrusted.

Processes of group 1 (Critical and trusted) can run in kernel space, and shared with processes of other groups.

Improvement Based On Liedtke's Work (Cont.)

This can improve performance for IPC in these conditions:

- Process of group 2, 3, 4 switches to process of group 1.
- Process of group 1 switches to other process in same group.
- Process of group 1 switches to other process out of group 1, and the target process is the previous process out of group 1 which switched to group 1 process.

In the above conditions, there is no TLB flushing in context switching for IPC. The performance of IPC can perform as well as Liedtke's original small address space scheme does.

In other conditions, TLB flushing must be performed in context switching.

Improvement Based On Liedtke's Work (Cont.)

Making processes running in kernel space references idea from Mach's co-located server .

- Processes of group 1 is trusted, which means with full testing, maximized stability and security, running in kernel space will not destroy ukernel.
- Processes of group 2 and 4 is untrusted, there is possibility for them to destroy kernel or each other. They can only run in separated address space other than kernel address space.
- Processes of group 3 is trusted, but they are non critical. It is unnecessary to put them running in kernel address space. Further more, make them run in user space contributes to kernel security risk minimization.

Improvement Based On Liedtke's Work (Cont.)

Critical missions can be served by natural L4 servers other than L4Linux.

- Fully tested, stable natural L4 servers are critical and trusted, can run in kernel space.
- Unstable natural L4 servers are critical and untrusted, still run in separate user address space.
- When unstable natural L4 servers are tested fully and verified to be stable and trusted, they can be moved to run in kernel space without recompiling ---- just relocating the program binary into kernel address space.
- when a trusted nature L4 servers are found unstable, they can be moved out of kernel address space --- just relocating the binary into separate user address space.

Improvement Based On Liedtke's Work (Cont.)

Review the two previous questions.

Can Liedtke's idea still work well under the new conditions ?

- Yes.
- But some modifications are needed.

Can we KEEP improving IPC performance on MICROKERNEL based system ?

- Yes.
- Only IPC involved with in kernel servers can keep improving performance. But it is better than no.
- Even better performance in conditions of x86_64 platform, multicore processors.

The implementation is still hardware concrete.

Methods Comparison

Comparison is made for in kernel servers vs. small address space.

Though the basic idea for them both are avoiding TLB flushing in context switch for IPC. There are several difference between them, including strong points and weak points.

Methods Comparison

In kernel servers vs. small address space

- Small address space only takes effect on x86 platform with segment switching. In kernel servers can work on both x86 and x86_64 platform.
- Small address space can run any process which can be placed into small address space. In kernel server can only run critical and trusted processes, but without size limitation of address space.
- Protection for small address space is performed by segments' base and limit. Protection for in kernel drivers is performed by privilege protection of page table.
- Migration between small address space and separate big address space is performed by modifying segments' base and limit. Migration between in kernel address space and user address space is performed by program address relocating.

Methods Comparison

In kernel servers vs. small address space (Cont.)

- Processes moved in/out small address space can continue to provide services without any interrupt. Processes migrated between in kernel address space and user address space have to be restarted, therefore continuous working need some trick.

Methods Comparison

Natural L4 servers vs. L4Linux single server

- Flexibility on scheduling policies
  For large scale system, for example platform with 8 multicore processors, 80 cores for each processor. In order to provide low latency services, L4Linux has to create a thread for a service on each core. Only think look at the interrupt handlers, for each core there are 32 handler threads for traps/exceptions and 15 for hardware IRQs at least. There are 47 * 80 * 8 = 30080 threads in L4Linux address space ! This is only for interrupt handlers, not including other system services.
  Huge number of threads in single process is unsuitable for flexible scheduling policies. E.g applying different scheduling policies on different system services or different virtual machine processes.
  Dividing a L4Linux server to multiple natural L4 servers can decrease number of threads in a process, providing fine-graininess for scheduling.

Methods Comparison

Natural L4 servers vs. L4Linux single server (cont.)

- Improved security
  Any bug in L4Linux will crash the whole process, including huge number for threads on large scale platform. Natural L4 server provides a single service, one of the server crashed will not interfere other servers.
  Only servers in kernel space can destroy other servers or ukernel.
- Improved realtime performance
  L4Linux is not designed for realtime applications. Implement system services in natural L4 servers can make preemptive scheduling easier, which sequently improves performance of realtime applications.

Key Points Of Implementations

Migration between kernel and user address space

In x86_64 platform, there is no segment based addressing mode. Non-local addresses in programs have to be relocated for migrating server process between kernel and user address space.

- Assign 4GB address space for each natural L4 server. This is assured by linking process.
- A address space region of the whole address space can be used for slots of in kernel servers (each slot has 4GB address space for x86_64 platform, or 2MB, 4MB, 8MB... in x86 platform.
- In x86_64 platform, a free list maintains all free slots for in kernel servers. In x86 platform, every slot size (2MB, 4MB, 8MB, ...) has a free list maintains the free slots of corresponding size. The free lists are used in slots allocating for in kernel server.
- Where the server will be relocated in kernel address space depends on the linear address of the allocated slot.

Key Points Of Implementations

Migration between kernel and user address space (Cont.)

- The only difference for a server running in kernel or user address space is the privilege level (user vs. supervisor) of page table mapped its address space region.
- In kernel space servers do not use any source code or library for ukernel. They continue to use normal libraries with relocating, but the libraries in kernel space can only be shared with other in kernel servers.

Key Points Of Implementations

Migration between different address spaces by processes group

- In x86_64 platform, the address space is large enough to hold all the processes running in one 64 bit address space.
- To improve the performance for process migration between different address spaces, processes can be migrated from user address space to kernel address space by processes group instead of one by one.
- Each process has LRU queues for last recent trusted processes which sent/receive messages to/receive to it. When a server is about to migrated into kernel address space, migrated all the servers in the LRU queues by group. This optimization can avoid relocating and TLB flushing for next time.
- Whether or not to flush TLB depends on how many page tables to be remapped. If the number of page tables is small than the threshold, use INVLPG instruction to invalid the TLB entry. If the number is large than the threshold, flush the whole TLB.
- Threshold can be determined in running time.

Key Points Of Implementations

Work both on x86 and x86_64 platforms

- Use different slot size for x86 and x86_64 platform. In x86 platform, the slot sizes are same to slot sizes of small address space. In x86_64 platform, size of all slots are 4GB (current size for experiment).
- Other platform dependent stuffs between x86 and x86_64 platform are unavoidable.

Difficulties To Be Overcome

- Run time relocation.
  For the time limitation, run time process relocation for migration in different address space can not be implement now. Use manual link instead, which does not matter for performance benchmark.
- Dynamic threshold determining for TLB flushing
  The threshold for TLB flushing has to be determined in processor initiation. Need to determine the threshold in a secure, simple and efficient method.
- No ready made test bed for benchmark
  No ready made 2nd generate opensourced microkernel based system implements most of the system services by natural server processes. Write a very basic microkernel system, with large numbers of dummy natural servers running in user or kernel space. Implement the idea of in kernel servers on this microkernel system and perform the benchmark based on this test bed.

Difficulties To Be Overcome (Cont.)

- Benchmark in an ideal environment
  The test bed is built for an ideal environment, therefore the result of benchmark is for ideal conditions.
  Can we persuade others by the benchmark from ideal environment ?

Thank You !

Any Comment Is Valuable To Me !