CS510 Languages and Low Level Programming: Portfolio submission, Topic 6

Due on May 13, 2016 at 11:59pm

Mark P. Jones Spring 2016

Konstantin Macarenco

Topic 6.

Describe the motivation, implementation, and application of microkernel abstractions for managing address spaces, threads, and interprocess communication (IPC).

Idea behind microkernel is to provide "policy free" environment, with no assumptions about services it will be running. All functionality of a traditional kernel is moved out, and implemented as a set of independent services. Microkernel provides only minimum set of abstraction over hardware, such as

- Address Spacing
- Inter-process Communication (IPC)
- Thread Management
- Unique Identifiers

Systems that use this approach will implement only the functionality needs to operate, have smaller footprint (embedded systems). All nonessential for the kernel functions, like network support, are implemented as services. Each service has it's own address space, communication with other services allowed only through the kernel. Only kernel functionality is allowed to run in privileged mode.

Advantages

- Robustness, a service crash doesn't affect any other part of the system.
- Easier maintenance.
- Enforces modular structure.
- Small size of Trusted Computing Base.

Disadvantages

- Performance loss due to high overhead of services interaction.
- Larger memory footprint (compared to monolithic OSs).
- Complicated process management.

Earlier versions of microkernel implementation suffered from high IPC overhead and did not look promising. L3/L4 (Jochen Liedtke) implementation proved that microkernel can be successful. Liedtke IPC design was 20 times faster than Mach's (fist generation of microkernel).

Implementation details

L4 implemented as a single linux server, and multiplexed tasks for user processes which means that each task is also a server for it's child tasks, basically acting as Virtual CPU.
On booting Linux server request memory from it's pager which maps physical memory in the server's address space from it's pager which maps physical memory in the server's address space, then acts as the pager for the user processes it creates.
Each task consists of Threads, memory objects and address space.

1. **Address Spacing**
   The L4 microkernel uses recursive address space model.
   Initial address space σ_0 represents all physical memory available. Each Threads basic memory operations are granting mapping and unmapping, which can be done only on the address space mapped (available to the process), e.g. Root process can map entire physical memory.
   Hardware interrupts are handled by user level processes, and triggered by the L4 kernel IPC. Each process is a server for it's children, and is responsible for handling page faults of the processes it created. When pagefault occurs, typically thread's pager creates new mapping and send it to faulty thread.
   Microkernel supports flexpages - simple pages that can be varied in size. In practice all pages larger than 4KB are just a collection of multiple 4K pages.

2. **Thread Management**

   The L4 thread management system tries to enforce processes hierarchy and hide existence of other services in the system.
   It is done by using thread multiplexing mechanism, with root being the first "thread". Each thread is as a server (root) for it's processes, and it can multiplex only it's resources.
   Each user process has associated Pager, Exception Handler and Scheduler threads, where pager is usually the user process's creator. Schedule threads can set scheduling parameters, like priority, time slice and quantum. Time slices can be donated to any other thread, or to kernel through donating time to nilthreads.
   Kernel Information Page (kernel version, host system info, address space layout info, system call entry points).
   User processes have User Thread Control Block (UTCB), which is used for communication with kernel.
   Threads can be referenced by thread id's. System defines global Thread ID, global interrupt ID, local thread ID. Ideally user processes should know only the thread ID's installed in UTCB table. Because of that global Ids are considered to be a bad practice - any thread can reference any other thread by global id, this potentially can lead to DOS attacks. Even if global thread ID is unknown it can be guessed (this is one of the problems seL4 tries to solve by using capabilities).

3. **Inter-process Communication (IPC)**
   Microkernel designed to restrict any process interaction except IPC. L3 and earlier implementations had much more complex IPC system than L4, which lead to high overhead.
   Even in L4 system calls are expensive, therefore should be avoided.
   This is done by:

   First, combining send and receive into a single system call. Before IPC system call thread loads message registers into the UTCB with a message to send, after IPC is complete sender thread is resumed, and it can read response from the receiver by reading UTCB registers.

   Second, If message is small enough to fit into CPU registers IPC call performance will be a lot faster, and most of the messages are small - %60 to %70 percent can be transfered just by using CPU registers. This development lead to significant performance boost.

   And last, In earlier microkernel generations Large message transfer was done by double copying message from sender to kernel and to receiver, which requires TLB flush. L4 avoids this by mapping target's area to receiver's memory space, temporarily sharing region of memory.