L02d: The L3 Microkernel Approach

Introduction:

- The SPIN and Exokernel approaches originated from the assumption that microkernel-based systems are inherently inefficient and has poor performance. That's because the popular microkernel-based OS back then, Mach-OS, was designed with portability as its most important goal.
- L3 microkernel approach was introduced to prove that performance is achievable with a microkernel-based OS.
- Microkernel-based OS structure:
  - The microkernel provides only core abstractions and has its own address space.
  - Many traditional OS services are implemented as processes on top of the microkernel. Each service has its own address space. These services run at the same privilege level as the user-level applications.
  - Communication between system services and user-level applications are done through the Inter-process communication (IPC), which is provided by the microkernel.
  - Potential of performance loss:
    1. Kernel-user switches: The border crossing cost related to the communication between the user-level applications and the system services.
    2. Address space switches: The cost of Protected Procedure Calls between system services across protection domains.
    3. Thread switches and IPC: The thread switches happen with Protected Procedure Calls need the intervention of the kernel, which adds also the cost of inter-process communication.
    4. Memory effects: These different calls move between different address spaces, so with each call we need to switch address space, so we lose the TLB contents and the cache contents (locality loss), which make them very expensive.

L3 Microkernel:

- The L3 Microkernel provides the core OS abstractions (Address space - Threads - IPC - UID).
- The basic argument behind L3 is that we can achieve an efficient microkernel OS if we maintained proper implementation.
- The difference in L3 Microkernel is that it enforces that each of the system services has to be in its own "protection domain" not necessarily in its own distinct address space.
- L3 Microkernel developers argued that we could construct an efficient microkernel-based OS if we know the features of the underlying platform.

- L3 Microkernel improvements:
  - Border crossing:
    1. L3 can perform a border crossing, including TLB and Cache misses, in only 123 processor cycles. Opposed to the 900 processor cycles that Mach-OS takes on the same HW (mainly because Mach doesn't have performance as its first priority).
    2. The process actually takes 107 processor cycle to execute the machine instruction themselves, which makes the 123 cycles that the L3 takes very close to the minimum time.
  - Address space switches:
    1. Whenever we switch from a specific address space to another, we might need to load the mappings of the new process to the TLB (perform TLB flush).
    2. This depends on whether the TLB has address space tags in addition to the normal virtual address tags. These TLBs are called "Address Space Tagged TLBs".
    3. In an Address Space Tagged TLB, whenever we create a TLB entry, we not only save the tag of the virtual space address, but also the PID of the process for which this entry is being created.
    4. In an Address Space Tagged TLB, we don't need to flush the TLB when we perform a context switch.
    5. This depends on the HW support. If the HW doesn't support address space TLB tagging, some suggestions can work:
       → x86 - PowerPC: These architectures provide segment registers to bound the range of virtual addresses that can be legally accessed by a running process. We can use these segment registers to provide protection domains.
       → This way we'll not need to flush TLB in case of context switch.
    6. If the protection domains are too large, and the HW doesn't support address space TLB tagging, then a TLB flushing is inevitable.
    7. If we're switching between large protection domains, the switching cost itself is not important, the cache and TLB costs (locality loss) will be dominant.
  - Thread switches and IPC:
    1. The explicit costs of the thread switching include saving all the volatile state of the processor in the thread context block.
    2. By construction, L3 shows that a microkernel is as competitive as SPIN and Exokernel in this area.
  - Memory effects:
    1. L3 suggests that if we have small protection domains, we should pack them together in the same HW space and enforce protection using segment registers.
    2. If the protection domains are large, the cost cannot be mitigated.

Mach-OS Border Crossings:

- Mach-OS focus mainly on portability, and it doesn't tailor any parts of the code to a specific HW. This results in a large footprint for the code base, which means less locality and more cache misses. This is the reason behind longer latency in border crossing.
- Portability and performance cannot be achieved together.

L3 Thesis for OS Structure:

- Minimal abstractions in the microkernel: The microkernel should only include the basic OS abstractions (Address space - Threads - IPC - UID). These abstractions are needed by any subsystem running above the microkernel.
- Microkernels are processor-specific in implementation: You should take advantage of whatever the underlying HW is providing. This will result in non-portability.
- L3 Microkernel is based on building processor-specific kernel and processor-independent abstractions at the higher layer of the OS stack.