L02c: The Exokernel Approach

Introduction:

- The kernel exposes HW explicitly to the OS extensions running above it.
- Exokernel processing steps:
  1. A library OS asks for a HW resource.
  2. Exokernel validates this request and bind the request to the HW resource (secure binding).
  3. After establishing the secure binding, Exokernel returns an encrypted key for the HW resource to the library OS.
  4. In order for the library OS to use the HW, it must present the encrypted key to the Exokernel.
  5. The Exokernel checks if the key is valid and validates whether the presenting library OS is the same one that requested the key or not.
  6. If the key is valid, the library OS gets access to the HW resource.
- Example: TLB entry:
  1. The virtual-to-physical address mapping is done by the library OS.
  2. "Secure Binding" is presented to the Exokernel.
  3. Exokernel puts the mapping into the HW TLB (privileged operation).
  4. Processes inside this library OS can use the TLB multiple times without Exokernel intervention.
- Establishing the binding is expensive (needs the kernel intervention) but using the kernel afterwards can happen at HW speed.
- Methods for implementing Secure Bindings:
  1. Hardware mechanisms (e.g. TLB entry).
  2. Software caching: A shadow TLB in the software data structure that is associated with each library OS.
  3. Downloading code into kernel: This is similar to SPIN extension, and it compromises protection. This is more dangerous than the SPIN approach, since in SPIN extensions are created at compile time and follow the rules enforced by the programming language. On the other hand, in Exokernel we're downloading arbitrary code into the kernel itself.

Default Core Services in Exokernel:

- Memory management:
  - What happens if a running process incurred a page fault?
    1. Exokernel will upcall the page fault to the library OS through a "Registered Handler".
    2. The library OS serves the page fault.
    3. The library OS performs the required mapping.
    4. This mapping is then presented to the Exokernel with the TLB entry.
    5. The Exokernel will validate the key and install the mapping into the HW TLB (privileged operations).

- Secure Binding:
  - The library OS is given the ability to drop code into the kernel itself.
  - This is intended to avoid border crossings and improve performance.
  - This greatly compromises protection.
  - The ability to create these extensions are restricted to only a trusted set of users to ensure protection.
- Software caching:
  - A method used by the Exokernel to implement Secure Bindings.
  - Since the address spaces occupied by two different library OSs are completely separate, a context switch requires flushing the TLB.
  - This produces a huge overhead for the new library OS to execute, because it will not find any of its addresses in the TLB.
  - The Exokernel creates a software TLB for each library OS to save a snapshot of the HW TLB of this library OS.
  - When a context switch happens, the Exokernel will dump the contents of the HW TLP to the SW TLP of the switched-from library TLB.
  - Then the Exokernel will pre-load the HW TLP with the contents of the SW TLB of the switched-to library OS.
  - This ensures that the library OS will find “some” of its mappings in the HW TLB.
- CPU scheduling:
  - The Exokernel maintains a linear vector of “time slots”, where each library OS gets a time quantum.
  - Each library OS should mark its time quantum at startup.
  - The Exokernel is scheduling the CPU according to this time vector.
  - The Exokernel doesn’t interfere with a library OS during its time quantum.

Revoking Resources:

- The Exokernel keeps track of which HW resources have been allocated to the different library OS’s.
- The Exokernel provides a “repossession vector” to each library OS to describe the HW resources it’s revoking.
- The library OS takes the corrective actions to clean up these resources before giving them back to the kernel.
- These corrective actions can also be done by the kernel if the library OS indicated this by “seeding” the kernel for “autosave” beforehand.

Conclusion:

- The Exokernel maintains a boundary between the different library OS's running on top of the kernel and the kernel itself, and also between the library OS's.
- To improve performance, the Exokernel provides the ability to "inject" code securely directly to the kernel.
- If the running threads performed any operation out of its normal scope (e.g. system call, page fault, exception, etc.), or an external interrupt stopped the thread, a trap happens, and the Exokernel informs the library OS that the thread is to be discontinued. After that, the Exokernel runs another library OS according to the time vector.
- To facilitate the above scenario, the Exokernel maintains a state of each library OS living on top of it (PE Data Structure).
- The PE Data Structure contains the different entry points for each library OS that correspond to the discontinuities that can happen while running a thread.