IPC Emulation Protocol

To emulate the seL4 IPC syscalls, we implemented two different protocols at the moment. One follows the seL4 IPC syscall semantic. The other one is used internally by the emulation framework. (e.g. Handshaking with the kernel in runtime setup routine). The IPC message layout is as follow:

Tag	ID	Reserved	Body
Word Size	Word Size	Word Size	Word Size * n_contextRegisters

Because in the real seL4 system, after traping into the kernel, all the user context registers are stashed in the tcb structure.
So we decided to pass an **`n * word size`** length message for each emulated IPC invocation, where **`n = n_contextRegisters`**. (**`n_contextRegisters`** is defined in the arch dependent **`registerset.h`**, on x86_64 it is **24**)

One challenge is how to deliver other message registers on the IPC Buffer? In the real seL4 system, as the kernel can accessing directly to the seL4 thread's IPC buffer. (explain in later slides). But for the emulation it's not quite easy to access the kernel emulator and the seL4 threads are running as different processes on Linux (explain in later slides), they have different address spaces. We need other process communication mechanisms.

- passing the entire IPC buffer on each emulated seL4 IPC invocation. (extra 8KB for a round trip of the IPC invocation, will be a horrible overhead for the IPC heavy applications!) ✗
- map the IPC Buffer as a shared memory. This approach is quite efficient and solved the problem quite well. As from the seL4 thread's view, it writes to the IPC buffer as usual,

however, the kernel emulator can access the seL4 thread's IPC Buffer directly as well. ✓