

Higher Half Kernel

From OSDev Wiki

It is traditional and generally good to have your kernel mapped in every user process. Linux, for instance (and many other Unices) reside at the virtual addresses `0xC0000000` - `0xFFFFFFFF` of every address space, leaving the range `0x00000000` - `0xBFFFFFFF` for user code, data, stacks, libraries, etc. Kernels that have such design are said to be "in the higher half" by opposition to kernels that use lowest virtual addresses for themselves, and leave higher addresses for the applications.

Advantages of a higher half kernel are:

- It's easier to set up VM86 processes since the region below 1MB is userspace.
- More generically, user applications are not dependent on how much memory is kernel space (Your application can be linked to `0x400000` regardless of whether kernel is at `0xC0000000`, `0x80000000` or `0xE0000000` ...), which makes ABI's nicer.
- If your OS is 64-bits, then 32-bit applications will be able to use the full 32-bit address space.
- 'mnemonic' invalid pointers such as `0xCAFEBADE`, `0xDEADBEEF`, `0xDEADC0DE`, etc. can be used.

Kernel Designs

Models

Monolithic Kernel
Microkernel
Hybrid Kernel
Exokernel
Nano/Picokernel
Cache Kernel
Virtualizing Kernel
Megalithic Kernel

Other Concepts

Modular Kernel
Higher Half Kernel
64-bit Kernel
Bare Bones

Initialization

To setup a higher half kernel, you have to map your kernel to the appropriate virtual address. How to do this basically depends on **when** you'd like your kernel to believe it's in the higher end, and **when** you set up paging.


Custom Bootloader

The easiest way is to load your kernel to any physical location you wish (for instance in the lowest 1MB) and prepare page tables that will perform the appropriate translation. Let's say you loaded your kernel at `0x00010000` to `0x0009FFFF` and want it to appear at `0xC0010000`, you could do the following:

- Pick 3 page-aligned (0x1000-aligned) addresses where you'll put your page directory and system tables. Make sure they are zeroed (*memclr* them or *memset* them to 0).
- Fill the lowest 256 entries of one table to set up Identity Paging for at least the BIOS and your bootloader (it's probably best to use 1:1 mapping for the entire lowest 1MB).
- In the other table, fill entry #0x10 (#16) with `0x00010003`, entry #0x11 (#17) with `0x00011003`, and so on (do this for every page your kernel has or needs).
- Fill entry #0x0 (#0) of the directory with the address of the first table (and make sure it's set to *present*).
- Fill entry #0x300 (#768) of the directory with the address of the second table (and make sure it's set to *present*).

When switching to Protected Mode, use this assembly example:

```
mov eax, physical_address_of_the_directory ; Get the physical address of  
mov cr3, eax ; ... and store it in CR3.  
mov eax, cr0 ; Get what's in CR0...  
or  eax, 0x80000001 ; ... enable protected mode and paging ...  
mov cr0, eax ; ... and put the new value back in CR0.
```



See Also

- I wrote a simple HigherHalf kernel (<http://forum.osdev.org/viewtopic.php?t=11160>)

Retrieved from "http://wiki.osdev.org/index.php?title=Higher_Half_Kernel&oldid=17151"

Category: Kernel

-
- This page was last modified on 30 November 2014, at 16:09.
 - This page has been accessed 39,101 times.