

Quick Guide to the Memory Manager Source#

Where is the source?#

- </services/system/memmgr>

What are the good bits?#

Functional Area	Related Files
virtual memory manager	services/system/memmgr/vmm*.*
physical allocator	services/system/memmgr/pa.c
memory manager data type implementations	services/system/memmgr/mm*.*
resource manager API to the memory manager	services/system/memmgr/memmgr*.*
CPU-specific implementation details	services/system/memmgr/<CPU>/*

Overview#

Though the OS is written in C, the memory manager implementation supports an object-oriented concept: [polymorphism](#). There is a global variable called *memmgr* that defines an API to the memory manager. The type of the object is [struct memmgr_entry](#), which defines a few data fields and a whole bunch of functions.

Throughout the code we make a lot of references to the *memmgr* variable. For example, when we need to change address spaces we call `memmgr.ospace()`.

But what's behind the *memmgr* variable? It could be any of a couple of different memory manager implementations. In practice, it's always the virtual memory manager (implemented in `services/system/memmgr/vmm*.*`) but it's possible that it could be the physical memory manager (`services/system/memmgr/pmm*.*`) or the runtime-executive memory manager (`services/system/ker/emm*.*` and `services/system/ker/smm*.*`).

When the system initializes, one of those memory managers is assigned to the *memmgr* variable (have a look at the `init_memmgr()` function at the end of [memmgr_init.c](#)). As I said, in practice it's always the virtual memory manager but the others are possible. But when you're looking at the kernel code and you see something like a call to `memmgr.foo()`, you can assume that `memmgr.foo()` is implemented with `vmm_foo()`.

For a more detailed look at the design of the memory manager, refer to [Virtual memory manager data structures](#) and [Virtual memory manager algorithms](#).