

Quick Guide to the Kernel Source#

The kernel and the [process manager](#) make up the operating system. They share the same address space. The kernel is non-threaded, and forces mutual-exclusion so that there is only ever one instance of it running, (even on SMP (small lie, see the SMP section)). The kernel "is entered" when anyone invokes a kernel call (or system call). The kernel runs in a privileged supervisor (or "ring0" in some architectures) state. The kernel only very rarely inhibits interrupts (and has an extensive and clever way of dancing out of the way of them). The most notable invoker of the kernel is the process manager or "[proc](#)".

Where is the Kernel Source?#

[/services/system/ker](#)

How do I checkout and build the source?#

Actually, one always builds the kernel and the process manager together:

- [checkout](#)
- [build](#)

Where are the good bits?#

Functional Area	Related Files
kernel public headers	services/system/public/sys
kernel internal headers	services/system/public/kernel kmacros.h kerproto.h externs.h
struct defs for thread, process, channel , interrupt ...	services/system/public/kernel/objects.h
low-level kernel data structures	nano_object.c nano_lookup.c kerext_query.c nano_query.c nano_vector.c nano_alloc.c
debug support, reading kernel internal data	kprintf.c cpu_debug.c kerext_debug.c nano_debug.c nano_kerdebug.c deb_rec.c
floating point	nano_fp_emu.c
kernel api, linkage kernel call entry, locking and exit(4)	kerext_* ker_* kerargs.h ker_call_table.c ker_ring0.c nano_spectet.c kerext_misc.c nano_misc.c asmoff.c

	module_list.S
kernel entrypoint, init, config	_main.c init_nto.c kgetopt.c kmain.c idle.c init_objects.c nano_conf.c
instrumentation, tracing	kertrace.h ker_trace.c kerext_trace.c nano_trace.c
interrupts, events, signals	ker_interrupt.c nano_interrupt.c nano_clock.c nano_smp_interrupt.c nano_event.c ker_signal.c nano_signal.c
magic	arm/kernel.S mips/kernel.S ppc/kernel.s sh/kernel.s x86/kernel.S
messaging, send, receive, reply	nano_message.c nano_xfer.c nano_xfer_msg.c nano_xfer_check.c nano_xfer_cpy.c ker_channel.c ker_connect.c ker_fastmsg.c ker_message.c
messaging, asynchronous	nano_asyncmsg.c
messaging, networked	ker_net.c
messaging, pulses	nano_pulse.c nano_xfer_pulse.c
MMU stuff cache control, address spaces, page faulting	kerext_cache.c walk_asinfo.c kerext_bind.c kerext_page.c kerext_stack.c
memory manager stubs(2)	smm_* nano_memphys.c
embedded memory manager(1)	emm_init_mem.c emm_mmap.c emm_munmap.c emm_pmem_add.c emm_vaddr_to_memobj.c nano_memphys.c
Minidriver support(6)	mdriver.c
mutex support	ker_sync.c
thread synchronization	nano_sync.c

power management stub	kerext_cpumode.c
process creation/destruction	kerext_process.c
security (uname,pw), rlimits	kerext_cred.c nano_cred.c kerext_limits.c
scheduler	ker_sched.c nano_sched.c nano_clock.c kexterns.c
Adaptive Partitioning Scheduler	see the source guide for aps
shutdown, reboot	kerext_reboot.c shutdown_nto.c
Symmetric Multiprocessing support(3)	nano_smp_interrupt.c smp_flush_tlb.c smp_get_cpunum.c smp_send_ipi.c services/system/smpswitch.h ker_nop.c
system page firmware/bios interface, key kernel globals	ker_sys.c nano_syspage.c
timers, clock	nano_clock.c ker_timer.c nano_timer.c ker_clock.c
thread creation/deletion	nano_thread.c ker_thread.c

Footnotes

(1) The embedded memory manager is only used when the OS is built as a stand-alone executive, i.e. without the process manager. QNX never ships it that way. So the `emm_*` files are generally not used. The nominal [memory manager](#) is in the process manager.

(2) The [memory manager](#) implementation is located in `proc`. It overrides these stubs in the kernel.

(3) Actually there is SMP support scattered throughout the kernel. Look for `#if defined(VARIANT_smp)`. Kernel modules, which aren't allowed to have compile variants, will test **NUM_PROCESSORS** or **runmask** instead.

(4) How we enter and exit the kernel is described [here](#).

(5) There is no footnote 5.

(6) Fixme: wth are minidrivers

Where the heck is the implementation of `KernelCall()`?

You won't find it by grepping. To find the implementation of a kernel call spelled like `CamelCaseKernelCall`, look for a function `ker_camel_case_kernel_call`. For example the `SchedCtl()` kernel call is implemented by `ker_sched_ctl()`. How the invocation sequence works is described [here](#).

What are the file naming conventions?#

filename pattern	Means
ker/ker_*	kernel entry file. Checks parameters and decides when to lock the kernel
ker/nano_*	low-level implementation, in-kernel state and in ring-0 or supervisor mode
ker/arm/* ker/mips/* ker/ppc* ker/sh/* ker/x86/*	processor specific code for ARM, PowerPC, Hitach SH-4, and Intel x86 processor architectures