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

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

| | |
|---|---|
| power management stub | kerext_cpumode.c |
| process creation/destruction | kerext_process.c |
| security (uname,pw), rlimits | kerext_cred.c<br>nano_cred.c<br>kerext_limits.c |
| scheduler | ker_sched.c<br>nano_sched.c<br>nano_clock.c<br>kexterns.c |
| Adaptive Partitioning Scheduler | see the source guide for aps |
| shutdown, reboot | kerext_reboot.c<br>shutdown_nto.c |
| Symmetric Multiprocessing support(3) | nano_smp_interrupt.c<br>smp_flush_tlb.c<br>smp_get_cpunum.c<br>smp_send_ipi.c<br>services/system/smpswitch.h<br>ker_nop.c |
| system page<br>firmware/bios interface, key kernel globals | ker_sys.c<br>nano_syspage.c |
| timers, clock | nano_clock.c<br>ker_timer.c<br>nano_timer.c<br>ker_clock.c |
| thread creation/deletion | nano_thread.c<br>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 seqence 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/*<br>ker/mips/*<br>ker/ppc*<br>ker/sh/*<br>ker/x86/* | processor specific code for ARM, PowerPC, Hitach SH-4, and Intel x86 processor architectures |