The Windows NT 3.1 Kernel's Source Tree

From a modern perspective, the compiling and linking of 32-bit code was still very primitive in 1993 when Windows NT 3.1 was released as the first Windows that's its own operating system. The 32-bit code for the Windows that ran on DOS had plausibly all been written in assembly language. The Device Driver Kit (DDK) for this variety of Windows was still years from acknowledging that 32-bit code for a Virtual Device Driver (VxD) in ring 0 might be written in C. Though the new Windows had a kernel, drivers and applications all in C from the start, the programming support was surely understood as primitive even from the perspective of the time. Notably, whatever capability the 32-bit C compiler yet had for 64-bit arithmetic was so recent that the Windows NT 3.1 kernel is written not to use it and the DDK for Windows NT 3.1 defines **LONGLONG** and **ULONGLONG** as **double** for the i386 processor, presumably to get 8-byte types with 8-byte alignment without yet depending on the compiler to have built-in 64-bit integers.

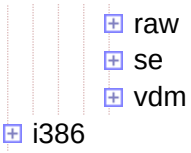
Something else that looks to have been relatively new in 1993 is the removal of debugging information from the executable to a separate symbol file. Pre-release builds of the Windows NT 3.1 kernel, as can be found on the Internet at websites that preserve what they call abandonware, have the debugging information in the executable as described by the Common Object File Format (COFF). Its removal to a separate file in the released Windows NT 3.1 looks to have been new enough that the .DBG files for both NTOSKRNL.EXE and NTKRNLMP.EXE are too primitive for Microsoft's linker to dump in any detail: feed them to **dumpbin /all** and the linker faults with an "Internal error during DumpDebugFile". Among their problems by modern reckoning is that they have zero for the **SectionAlignment** in the **IMAGE_SEPARATE_DEBUG_HEADER** and their **IMAGE_DEBUG_DIRECTORY** for **IMAGE_DEBUG_TYPE_MISC** is misformed. Still, they are easily enough inspected by sight. They can even be edited so that Microsoft's linker parses them without faulting.

What inspecting them will get for you most obviously is the matching of Microsoft's names to the addresses of routines and variables. Of interest here is that the COFF symbol table also has **IMAGE_SYM_CLASS_FILE** records for the source files. In these particular .DBG files, the names are observed to be fully qualified for source files in C but only relative for assembly-language source files. For both the debugger and the reverse engineer, it is straightforward to determine which routines and variables are in which source files and to put it all together for a map of where Microsoft had these source files at the time of compilation.

Thus did Microsoft from the start publish its source tree for the Windows kernel—not the source code or the source files, of course, but the directory structure in which those files were organised. For the tree below, I don't go as far as showing which routines and variables are in which source files, just the tree of source files in directories. If you browse this page with scripts enabled, then you should be able to expand the branches that interest you and collapse those that don't.

- D:
  - nt
    - private
      - crt32nt
      - lsa
      - ntos (but ntosmp for multi-processor build)
        - brtl
        - cache
        - config
        - dbgk

- ex
- fsrtl
- init
- io
- kd
- ke
  - i386
    - abiosc.c
    - allproc.c (only in multi-processor build)
    - apcuser.c
    - biosc.c
    - dmpstate.c
    - exceptn.c
    - flushtb.c
    - gdtsup.c
    - i386init.c
    - intobj.c
    - iopm.c
    - kernlini.c
    - ldtsup.c
    - mpipi.c
    - thredini.c
    - trapc.c
    - vdm.c
  - apcobj.c
  - apcsup.c
  - balmgr.c
  - bugcheck.c
  - config.c
  - debug.c
  - devquobj.c
  - dpcobj.c
  - dpcsup.c
  - eventobj.c
  - kernldat.c
  - kiinit.c
  - miscc.c
  - mutexobj.c
  - mutntobj.c
  - procobj.c
  - profobj.c
  - raisexcp.c
  - reqint.c
  - semphobj.c
  - statsobj.c
  - thredobj.c
  - thredsup.c
  - timerobj.c
  - timersup.c
  - wait.c
  - waitsup.c
- krtl
- lpc
- mm
- ob
- ps



Presumably, the i386 directory for assembly-language files is in each case a subdirectory of one of the many subdirectories of D:\nt\private\ntos, but although the particular subdirectories for most might be inferred confidently, I don't see how to do so for all, or for any except with information from elsewhere (whether assumed or known), and so I stick just to the names as recorded in the symbol files.

Two files are named only as i386\raise.asm. They are distinct. They each define only one routine: **ExRaiseException** and **RtlRaiseException**. These do eventually become exported functions but in version 3.10 they are internal routines. Almost certainly, these assembly-language source files are in the ex and krtl subdirectories.

For source files in C, the symbol files provide full pathnames. They look to be broadly in two sets. Very many are in a subtree that is specifically for the kernel. More than a few others, notably those for the C Run Tme (CRT), look to be shared with other modules, possibly even for execution in user mode. Of the source files that are specifically for the kernel, all but one contribute to both the single-processor and multi-processor kernel. How these source files are compiled for the variant kernels is unclear. To go from the pathnames alone, each exists in two places, one somewhere under d:\nt\private\ntos, the other in the same place but under d:\nt\private\ntosmp instead. It's not credible that the two are maintained independently. One, if not both, is surely a copy made just for the build. Later versions, certainly as soon as version 3.51, are known to compile each source file at the one location, under a subdirectory named ntos, to produce object files in different subtrees for single-processor and multi-processor builds. For an example, put aside the complication of processor-specific source files, such as seen above in subdirectories named i386. Whatever directory *source*.c is in has subdirectories named um and mp. The source file is built from the subdirectory as ..*source*.c to produce an object file objfre\i386*source*.obj. Presumably, this scheme was yet to be devised when version 3.10 was built. The makefile in the Windows NT 3.1 DDK allows only for object files in subtrees for different processors. In the simple example just given for later versions, *source*.c is compiled in-place to produce obj\i386*source*.obj. A possibility for how version 3.10 accommodates single-processor and multi-processor builds is that the ntos subtree is the source files' home and the ntosmp subdirectory receives copies from which to build the multi-processor kernel.

This page was created on 24th July 2020 but was not published until 3rd November 2020. It was last modified on 3rd December 2020.