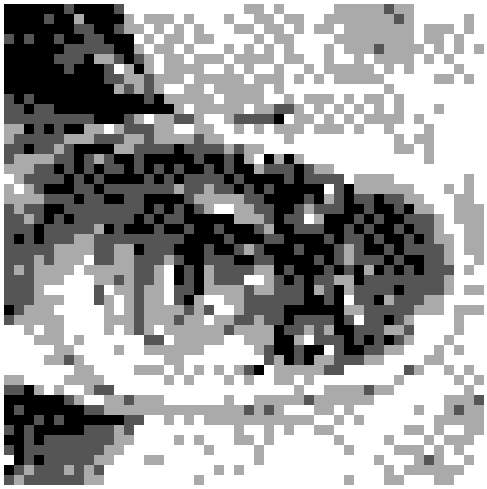
forsyth Fri Feb 5 11:12:02 GMT 2010



Inferno Ports: Hosted and Native

Vita Nuova
27 April 2005
Revised 22 January 2007

Inferno is a portable environment, encompassing operating system, languages, virtual machine and the tools required to build it. This section briefly summarises the state of the ports and compilers included in this release. Directory names are relative to the root of the Inferno release tree, unless otherwise specified by the context.

All components are built using the program `mk`, based on 'make'-like recipes found in the `mkfile` in each source directory throughout the Inferno tree. `Mk` is described by the manual page *mk*(10.1) in Volume 1; a more tutorial discussion, including a summary of differences with Unix *make*, can be found in *Maintaining Files on Plan 9 with Mk* by Hume and Flandrena, reprinted in this volume. The source for `mk` itself is included in `utils/mk`. It is included ready-made in the full and source-only distributions, to make life easier. It must be compiled manually only on the initial port to a new host environment; instructions for that are given below.

1. The C compilers

An unusual property of the compiler suites used to compile native Inferno is that there is no difference in configuration or content between a 'compiler' (compiling on the same system and processor type as the target) and a 'cross compiler' (compiling on a host that differs from the target), even when the host operating systems are quite different. Indeed, in their ancestral home, Plan 9, it is the default action to compile instances of all compilers for all possible target architectures, as a matter of course.

The main difference between this suite and the original Plan 9 suite is that all Plan 9 C extensions have been eliminated from the compiler's own source, allowing it to be compiled on environments that accurately support ANSI C and a few necessary Posix functions.

The source for the compilers is found in subdirectories of `utils`. The compilers are named as follows:

`0c`	MIPS compiler for 64-bit little-endian R4000 MIPS (or '`spim`')
`1c`	68000 compiler, usable with the Motorola Dragonball
`2c`	680x0 compiler for x >= 2
`5c`	ARM compiler
`6c`	AMD64
`8c`	Intel x86 compiler, for x>2
`kc`	Sun SPARC compiler
`qc`	PowerPC compiler
`vc`	MIPS R[234]000 in 32-bit big-endian mode

The compilers share components, compiled into a library from source in the directory `utils/cc`. The corresponding assemblers and linkers are found in similarly named directories: `2a` and `2l` are the assembler and linker for use with `2c` for instance. Note that this suite is unusual in that the compilers and assemblers produce a binary assembly language that is finally converted to machine code by the linker. The assembler is used only to write machine-language assist for the operating system, or a run-time routines using instructions not accessible from C, and is not used by the compiler. See the paper ''Plan 9 C compilers'' by Ken Thompson, reprinted in this volume.

With the exception of the 68000 compiler, all the compilers have been used extensively to compile Inferno, and most have been used to compile Plan 9 and all its applications; and we have found them solid. The 68000 compiler was used to attempt a port of Inferno to the Motorola Dragonball (in the Palm Pilot). It is included here in case someone wishes to have another go. We have no experience with it.

The ARM compiler `5c` supports the ARM (Strongarm, PXA) architecture; the related compiler `tc` generates ARM's Thumb instructions instead. The output of both can be linked together by the ARM loader `5l` to achieve ARM-Thumb interworking. `5c` has been used to generate code for the StrongARM SA110 and SA1100 processors (the primary targets for native Inferno for many years). The code generated was greatly improved by Richard Miller. The floating-point support is adequate for C programs: the compiler generates ARM floating-point instructions, as implemented on the ARM7500 but not on the Strongarm, where they must be emulated.

The PowerPC compiler supports the 32-bit PowerPC architecture only; it does not support either the 64-bit extensions or the POWER compatibility instructions. It has been used for production operating system work on the 405EP, 603, 70x, 821, 823, and 860. On the embedded processors such as 405 and 8xx floating-point instructions must be emulated. Instruction scheduling is not implemented; otherwise the code generated is similar to that for the other load-store architectures. The compiler makes little or no use of unusual PowerPC features such as the counter register, several condition code registers, and multiply-accumulate instructions, but they are sometimes used by assembly language routines in the libraries. The compiler does replace explicit comparisons by condition-setting instructions. Its run-time conventions are more efficient than those of the PowerPC ABI.

2. Applications

Dis object files are portable across all variants of Inferno, hosted and native. There need be only one copy of the Dis files to serve many different versions of Inferno; they need not be rebuilt for each platform and can be shared by different types of host. Limbo insulates the programmer from all details of the particular processor, including byte-ordering, and consequently the applications themselves are portable.

The source for the applications is found in subdirectories of `appl`: `appl/cmd` holds the source for most command line applications (that use no graphics); `appl/wm` contains the source for most applications that run under *wm*(1); `appl/svc` contains the source for various system services and file servers; `appl/mux`, the source for the interactive television demo *mux*(1); `appl/charon`, the source for the Charon web browser; and `appl/acme` the source for Acme written in Limbo.

The `mkfile` in each directory can currently only be used by an instance of `mk` running *outside* the Inferno environment, under the host operating system. This complicates its use with *acme*(1), normally requiring the use of the *os*(1) command. In a few cases, there is a `mashfile` that can be used by *mash-make*(1) to build a Limbo application from within Inferno (native or hosted). A consistent approach to building applications both inside and outside Inferno is being developed. In any case, the resulting Dis files are portable once produced.

3. Hosted Inferno (emu)

There are currently four main variants of hosted Inferno: Plan 9, Unix (and clones), MacOS X and Windows. The source is held in directory `emu`, with a subdirectory for each hosted platform: `FreeBSD`, `Irix`, `Linux`, `MacOSX`, `NetBSD`, `Nt` (for all Windows platforms, including the Internet Explorer plug-in), `Plan9`, `Solaris`, and so on. Each platform directory has a `mkfile` and one or more configuration files of the form described by *config*(6). An executable for a particular host type is built on that host type, using the host's own command interpreter, not under Inferno. Move to the `emu` subdirectory appropriate to that host, ensure the command interpreter's path variable includes the directory containing the Inferno `bin` directory for that host (eg, `/home/inferno/Solaris/sparc/bin`), and run `mk`.

Like the native kernels `emu` relies on several auxiliary libraries (the source of which it often shares with the native kernels). Emu itself is built by the `mkfile` in the `emu` subdirectory containing the platform-specific source for the host platform. Each library has its own `mkfile`; the various components are made in the right order by the `mkfile` at the root of the Inferno tree. The `mkfile` for each platform will also invoke `mk` recursively to make the appropriate libraries for a given configuration.

The Unix emu variant generally is covered by 'POSIX' (with common extensions) but each Unix port has one file that differs considerably for each port, namely `emu/`*platform*`/os.c`, the differences corresponding to the different ways under Unix of implementing kernel-scheduled threads efficiently.

There are working emu versions for FreeBSD/386, Irix/mips, Linux/386, NetBSD/386, MacOSX/386, MacOSX/power, Plan 9, Solaris/sparc, and Windows (NT, 2000 and Explorer plug-in). Each platform typically uses mechanisms specific to the host operating system to implement Inferno's internal thread/process structure. POSIX threads have often been found to be insufficient (poorly implemented) on some platforms, and if so are avoided. See `kproc` in `emu/*/os.c`.

Source is included for ports to HP/UX (S800 architecture), Solaris/386, and Unixware, in case someone wishes to take them up now, but we have not determined their fitness.

The Plan 9 hosted implementation is unusual in that it supports several processor types: `386`, `mips`, `power` (Power PC) and `sparc`. Furthermore, all versions of `emu` can be built on any processor type, in the usual way for Plan 9.

Otherwise, as distributed, `emu` for a platform can only be built when running on that platform.

One unusual variant makes the whole of Inferno a plug-in for Microsoft's Internet Explorer, giving the same environment for Inferno applications running in an HTML page as is provided by hosted or native Inferno. That is, there is not a distinct 'applet' environment with special programming interfaces. The source for the various plug-in components is found in `/tools/plugin` and `/usr/internet` within the Inferno tree; they use the version of *emu* defined by the configuration file `/emu/Nt/ie`.

All the libraries and executables can be built in a tree containing only the source code. To do that for a supported variant of hosted Inferno, on Unix or Plan 9, do the following in the root of the Inferno tree:

1 Edit `mkconfig` to reflect your host environment, specifically ROOT (which must be an absolute path name), SYSHOST and OBJTYPE. The comments in the file should help you choose.

2 Run `makemk.sh` to rebuild the `mk` command, which is used to build everything else.

3 Set `PATH` (or `path` on Plan 9) to include the `bin` directory for the platform, which will now contain the `mk` binary just built. On Unix, export `PATH`.

4 Then `mk nuke` to remove any extraneous object files.

5 Finally, `mk install` to create and install the libraries, `limbo` compiler, `emu` for hosted Inferno, and auxiliary commands. The rules do that in an order that ensures that the commands or libraries needed by a later stage are built and installed first. (Note that a plain `mk` will not suffice, because it does not put the results in the search path.)

Doing something similar on Windows or Plan 9 currently requires the executable for `mk` to be available in the search path, since there is no equivalent of `makemk.sh`. Otherwise the procedure is the same. On Plan 9, of course, the host system's normal version of `mk` should be adequate.

4. Native Inferno

As with the different versions of emu, once the native kernel is running, all applications work straight away; the same applications are used in native and emulated mode, subject to suitable devices being available. Because the portable compiler suite is used to compile native kernels, and those compilers are automatically cross-compilers, all native Inferno implementations can be built on any host platform. Furthermore, the build procedures and resulting object files are the same.

Early ports in 1996 were made by Bell Labs to an internal device based on the AMD 29000, an early ARM-based 'network computer', and Intel-based PCs. Between 1997 and 1999, Lucent concentrated mainly on the Strongarm platform (SA1100), for various Digital/Intel development boards, and especially several 'web phones', including the Sword Webphone Reference Design. It also undertook ports to other devices for experiment, or under contract.

Vita Nuova Limited also ported the system, both for its own purposes and under contract to Lucent. Targets included a small 386-based Internet device, a set top Internet box using the PowerPC 603e, a digital television set top box with a Strongarm SA110 and a Teralogic TL750 graphics chip, the USR/3Com Edgeserver (in a chassis containing various types of line card), various boards based on the PowerPC 823/821/860, many different configurations of IBM PC, and a Ziatech Pentium-based VME crate.

Distribution of most previous and existing ports is restricted by the terms on which they were undertaken, or because they were ports of older Inferno releases and not kept up to date. We have included the following as examples in this distribution.

The StrongARM kernel

The source for the StrongARM kernels is split across several directories. The directory `os/sa1110` contains all code that is generally architecture-specific but platform-independent. Other directories contain platform-specific code: `os/cerf1110` for the Intrinsyc Cerfcube1110, and `os/ipaq1110` for the Compaq (as it then was) IPAQ H3650. Earlier Webphone ports are tied to hardware that is not generally obtainable and the ports to those platforms included some software (notably modem software) that cannot generally be distributed.

There is also a preliminary port to the ARM-based Intel XScale. The code common to PXA implementations is in `os/pxa`. The initial platform was the Intrinsyc Cerfboard 250; its code is in `os/cerf250`. A port to the Gumstix (see `www.gumstix.com`) is in progress.

The platform's own bootstrap is used in all cases. On the IPAQ, the Linux bootloader from Compaq (HP) Research must be loaded onto the device first, following instructions given at `www.handhelds.org`. See the `README` file in each `os` source directory for details.

Other ARM-based processors to which Inferno has been ported include the ARM-7 evaluator kit (see `os/ks32`), although its memory is tight, and the TI925 including the TI OMAP. The latter two ports were to proprietary TI925 implementations, and have not been included, but there is a body of code common to all such platforms that could be made available if that were useful.

The PowerPC kernel

The directory `os/fads` contains the port of Inferno to the MPC8xx FADS development board. It has been used with the MPC821, MPC823 and MPC860 processors. It uses code common to MPC8xx processors, found in `os/mpc`. The interface to the CPM is provided by `cpm.c`. There are drivers for the real time clock, flash devices (including a Flash Translation Layer driver), and communications controllers in Ethernet, UART, and IrDA mode (see `etherscc.c` and `devuart.c`). The IrDA has been used for 9P transport between a FADS board and an IBM Thinkpad 560. The file `screen.c` drives an 8-bit per pixel LCD (TFT) display panel. A sample interface to the on-chip video device of the MPC823 (only) as wired on the FADS board using auxiliary chips can be found in `devvid.c`. The York Electronics Centre developed a touch panel for us, connected using SPI; the driver is `devtouch.c`, and could be adapted for similar devices.

The bootstrap program for the FADS board is in `os/boot/mpc`, loosely derived from an older version of `os/boot/pc`. It is initially converted to S records that are loaded into flash by MPC8BUG from a PC, and thereafter the images of the boot and kernel images can be updated using the flash devices provided by the system itself, and the utility programs `qconfig.b` and `qflash.b` in `appl/cmd/mpc`.

Another port is to the Brightstar Engineering ip-Engine containing an MPC823 and an Altera FPGA. See `os/ipengine`. It uses common code from `os/mpc`. The device driver that loads the FPGA is in `devfpga.c`; see *fpga*(3) for the interface and *fpgaload*(8) for a command to do it. See the `README` file for information on loading the kernel into the flash.

The most recent PowerPC port is to the IBM 405EP, and more specifically to the Intrinsyc Cerfcube 405EP. The source for that port is in `os/cerf405`; lacking another 405EP platform for reference, the source code has not yet been split into that common to all 405EP implementations and that specific to the Cerfcube, although that would be easy to do.

The x86 kernel

The `os/pc` directory contains the components for ports to 386, 486 and Pentium class machines. The main difficulty is device support: in particular only a limited set of Ethernet and graphics cards is supported. We have used mainly the 3Com and Intel 82557 drivers. A 'generic' PC port is included that has a graphics driver that should run on systems that provide a VESA BIOS mode.

We have a (slow) floating-point emulator for the 386 found in `os/pc/fpi387.c`; code to invoke it in trap can be provided on request.

The source for the PC bootstrap program `9load` is in `os/boot/pc`. It is simply a copy of the current Plan 9 PC bootstrap program, with slight modifications to allow it to be compiled on many host systems.

The Javastation 1 kernel

The directory `os/js` has the first port to the Sun Javastation 1. It was done by Tad Hunt and Eric Van Hensbergen in a matter of days to demonstrate Inferno at Java One in 1997. It boots over the net using TFTP. Javastations being a bit thin on the ground now, it is unlikely to be directly usable unless you can find one second hand (you might find a Javastation 2 coffee pot, but that is slightly different again). That is a pity, because the machine was quite usable running Inferno and Limbo applications, often surprising those used to the Java-based offering on the same platform. It is included as an example of a micro-SPARC port. Beware that `screen.c` has not yet been converted for Fourth Edition graphics (partly because we no longer have a suitable device for testing).

5. Supporting tools

The `utils` directory also contains ANSI C versions of other components of the Plan 9 development suite, such as `nm`, `ksize`, `ar`, and of course the `acid` debugger. Most rely on `libmach`, a suite of functions forming a library to handle the various object and executable files in one place.

Some other utilities give a portable way to express some of the kernel build scripts: `sed`, `test`, `rm`, and `mkdir`. On Plan 9, `mk` and kernel build scripts use Plan 9's own shell, *rc*. On Unix systems, they use *sh*. On Windows, a version of Plan 9's *rc* has been ported to reduce the number of variants to two, and keep the system self-contained; its source is in `utils/rcsh` and installs as `rcsh.exe`.