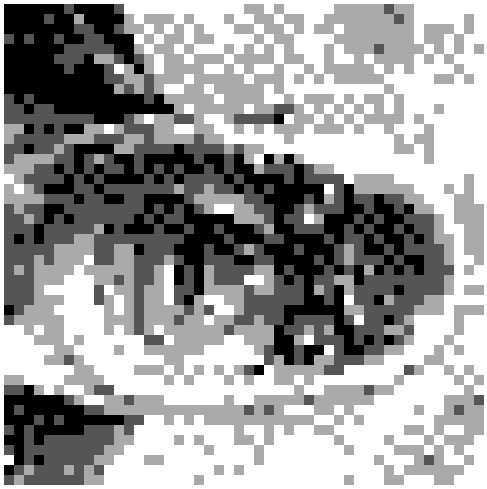
forsyth Mon Jun 29 10:03:03 BST 2009



Installing the Inferno Software

Vita Nuova
support@vitanuova.com
12 June 2003

Inferno can run as either a native operating system, in the usual way, or as a *hosted* virtual operating system, running as an application on another operating system. This paper explains how to install Inferno from the distribution media to a hosted environment and how to configure the system for basic networking.

Inferno can run as a hosted virtual operating system on top of Plan 9, Unix or Windows. In this paper, the term *Unix* is used to cover all supported variants, currently FreeBSD, Linux, HP/UX, Irix and Solaris, and the term *Windows* covers Microsoft Windows (2000, XP, and Vista).

1. Preparation

You should ensure at least 150 Mbytes of free space on the filesystem. The installation program will copy files from the distribution CD to a directory on the filesystem called the *inferno_root* directory. You can choose the location of this directory. If you are installing to a multiuser filesystem outside your control a subdirectory of your home directory might be most sensible. If you plan to share the Inferno system with other users then common choices for *inferno_root* are `/usr/inferno` on Unix and Plan 9 systems, and `c:\inferno` on Windows systems. Where these appear in examples in this paper you should substitute your own *inferno_root* directory.

- **Step 1a: Choose the *inferno_root* directory.**

Ensure that the user who will run the installation program has appropriate filesystem permissions to create the *inferno_root* directory and files and subdirectories beneath it.

2. Copying Files

On all platforms the files will be owned by the user doing the installation, except for installation onto a FAT file system (eg, on Windows), where the files appear to be owned by `Everyone` because FAT does not record ownership.

- **Step 2a: Insert the distribution CD into the CD drive.**

On Unix and Plan 9, mount the CD to a suitable location on the filesystem, call this location *cd_path*. On Windows, note the drive letter of the CD, call this drive letter *cd_drive*. The files will be copied by an Inferno hosted installation program which runs directly from the CD. The directory `/install` on the CD contains an installation program for each supported platform – a shell script for Unix and Plan 9 and an executable for Windows. The Plan 9 install script is called `Plan9.rc` and determines the CPU type from the environment variable `cputype`. The Unix install scripts all have names of the form *host_os*`-`*host_arch*`.sh` where *host_os* will be one of: `FreeBSD`, `Linux`, or `Solaris` and *host_arch* will be one of: `386`, `mips`, `power` or `sparc`. Most platforms offer just the one obvious combination. The Windows installation program is called `setup.exe`; it is used on all varieties of Windows. The next step describes how to begin the installation by running the program that corresponds to your host system.

- **Step 2b: Run the installation script.**

The installation program will copy files from the CD to the filesystem. The Windows installation program will also create registry entries and add an Inferno item to the Windows *start* menu. On Plan 9, run

`rc` *cd_path*`/install/Plan9.rc` *inferno_root*

Where *inferno_root* is the path to the chosen Inferno root directory. The CPU architecture will be inferred from the environment variable `cputype`. On Unix, run

```
sh cd_path/install/host-os-host_arch.sh inferno_root
```

Where *host_os* is the Unix variant name (`FreeBSD`, `Irix`, `Linux` or `Solaris`). *host_arch* is the CPU type (eg, `386`), and *inferno_root* is the path to the chosen Inferno directory. On Windows, run

```
cd_drive:\install\setup.exe
```

The Windows installation program will ask you to choose the location of the installation directory on the hard disk.

On all platforms, a copy of Inferno on the CD will install from various installation packages on the CD to the *inferno_root* subtree on the filesystem. On any platform it installs support for all.

Inferno is now installed, but it needs to be configured for your site. The process acts as a quick tour of parts of the system. The main tasks are to add local parameters to the network data base, and to set up the authentication system. If you are going to run Inferno standalone, for instance to experiment with Limbo and the file serving interface, most of what follows can be deferred indefinitely. It is still worthwhile skimming through it, because the first few sections tell how to start up Inferno with correct parameters (eg, root directory and graphics resolution). (A configuration program that runs under the window system would be more convenient, and fairly easy to do, but that has not yet been done.)

3. Running Inferno

Inferno host executables are all kept in a single directory corresponding to the combination of host operating system and CPU architecture – the Inferno `bin` directory.

```
inferno_root/host_os/host_arch/bin
```

(On Windows the path might need `\` not `/` of course.) That directory can be added to the search path of the host system's command interpreter, and that process will be described first, although as discussed later one can use a script instead and that is sometimes more convenient. (Of course, the script will still need to refer to that directory.)

Plan 9: Plan 9 users should add a line to their `lib/profile` file that binds this directory after their `/bin` directory.

```
bind -a /usr/inferno/Plan9/$cputype/bin /bin
```

The bind is done after the existing *bin* directory to avoid hiding the existing Plan 9 compilers. If, at a later stage, you build either the hosted or native Inferno kernels for ARM or StrongARM you should ensure that the Inferno compilers are used rather than the Plan 9 compilers, since they differ in the implementation of floating-point instructions (the Plan 9 ARM suite uses a byte order that is more plausible than the order ARM dictates but therefore wrong). That difference is likely to be resolved at some point but it has not yet been done.

Windows: The *host_os* is always `Nt` (even for Windows 2000, XP or Vista) and *host_arch* is always `386` and the installation program will create an entry on the *start menu* to invoke Inferno. For Unix systems or Windows systems in which Inferno will be started from a command shell, the environment variable `PATH` should be set to include the Inferno `bin` directory. For Windows NT and Windows 2000 modify the `Path` environment variable through *Control Panel -> System -> Environment*. A temporary change can be made in the `CMD` interpreter using:

```
PATH=c:\inferno\Nt\386\bin;%PATH%
```

where `c:\inferno` is assumed to be the location of the Inferno distribution.

If you are using an MKS or Cygwin Unix-like shell environment, you might instead set:

```
PATH="c:/inferno/Nt/386/bin;$PATH"
```

and export it if necessary.

Unix: For Unix systems, for `sh` derivatives, the environment variable `PATH` should be set to include the Inferno `bin` directory. This might be done in your `.profile` file by adding a line like

```
PATH="/usr/inferno/Linux/386/bin:$PATH"
```

Don't forget to ensure that `PATH` is exported. You may need to log out and back in again for the changes to take effect.

- **Step 3a: Start Inferno.**

Hosted inferno is run by invoking an executable called *emu*.

On Windows, select the Inferno option from the *start menu*. This will invoke *emu* with appropriate arguments to find its files in *inferno_root*. If you need to change any of the options passed to *emu* when invoked from the *start menu* you need to do this by clicking the right mouse button on the Windows task bar and choosing *Properties -> Start Menu Programs -> Advanced* to modify the shortcut used for Inferno. For Unix and Plan 9, you will need to tell *emu* where to find the Inferno file tree by passing it the `-r`*rootpath* command line option. For example

```
emu -r/usr/john/inferno
```

Without the `-r` option it will look for the file tree in `/usr/inferno` on Plan 9 and Unix and, when invoked from the command line on WIndows, the default is `\inferno` on the current drive. (The Windows start menu by contrast has already been set to use the right directory by the installation software.)

When using graphics, *emu* will use a window with a resolution of 640 x 480 pixels by default. To use a larger resolution you will need to pass *emu* an option `-g`*Xsize*`x`*Ysize* on the command line. So, for example, to invoke *emu* as above but with a resolution of 1024 x 768 pixels the full command line would be

```
emu -r/usr/john/inferno -g1024x768
```

When invoked in this way *emu* displays a command window running the Inferno shell `/dis/sh.dis`. To avoid typing the command line options each time you invoke *emu* you can store them in the environment variable `EMU` which is interrogated when *emu* is started and might as well be set along side the `PATH` environment variable if the same configuration options are to be used on each invocation.

```
set EMU="-rd:\Documents and Settings\john\inferno -g1024x768"
```

for Windows.

```
EMU=(-r/usr/john/inferno -g1024x768)
```

for Plan 9, and

```
EMU="-r/usr/john/inferno -g1024x768"
```

for Unix. An alternative to using the `EMU` environment variable is to place the correct invocation in a script file (or batch file, for Windows) and invoke that instead of running *emu* directly. It is important to note that for Windows the `-r` option also serves to indicate both the drive and directory on to which the software has been installed. Without a drive letter the system will assume the current drive and will fail if the user changes to an alternative drive. Once the environment variables or scripts are set up, as described above, invoking plain

```
emu
```

or the appropriate script file, should result in it starting up Inferno's command interpreter *sh*(1), which prompts with a semicolon:

```
;
```

You can add a further option `-c1` to start up *emu* in a mode in which the system compiles a module's Dis operations to native machine instructions when a module is loaded. (See the *emu*(1) manual page.) In *compile* mode programs that do significant computation will run much faster. Whether in compiled or interpreted mode you should now have a functional hosted Inferno system. When Inferno starts the initial `/dis/sh.dis` it reads commands from the file `/lib/sh/profile` before becoming interactive. See the manual pages for the shell *sh*(1) to learn more about tailoring the initial environment.

The semicolon is the default shell prompt. From this command window you should be able to see the installed Inferno files and directories

```
lc /
```

The command *lc* presents the contents of its directory argument in columnar fashion to standard output in the command window.

```
; lc /
FreeBSD/     Unixware/    icons/       libkern/     man/         prof/
Hp/          acme/        include/     libkeyring/  mkconfig     prog/
Inferno/     appl/        keydb/       libmath/     mkfile       services/
Irix/        asm/         legal/       libmemdraw/  mkfiles/     tmp/
LICENCE      chan/        lib/         libmemlayer/ mnt/         tools/
Linux/       dev/         lib9/        libtk/       module/      usr/
MacOSX/      dis/         libbio/      licencedb/   n/           utils/
NOTICE       doc/         libcrypt/    limbo/       net/         wrap/
Nt/          emu/         libdraw/     locale/      nvfs/
Plan9/       env/         libfreetype/ mail/        o/
Solaris/     fonts/       libinterp/   makemk.sh    os/
;
```

Only the files and directories in and below the *inferno_root* directory on the host filesystem are immediately visible to an Inferno process; these files are made visible in the root of the Inferno file namespace. If you wish to import or export files from and to the host filesystem you will need to use tools on your host to move them in or out of the Inferno visible portion of your host filesystem (see the manual pages *os*(1) and *cmd*(3) for an interface to host commands). (We plan to make such access direct, but the details are still being worked out.) From this point onwards in this paper all file paths not qualified with *inferno_root* are assumed to be in the Inferno namespace. Files created in the host filesystem will be created with the user id of the user that started *emu* and on Unix systems with that user's group id.

4. Setting the site's time zone

Time zone settings are defined by files in the directory `/locale`. The setting affects only how the time is displayed; the internal representation does not vary. For instance, the file `/locale/GMT` defines Greenwich Mean Time, `/locale/GB-Eire` defines time zones for Great Britain and the Irish Republic (GMT and British Summer Time), and `/locale/US_Eastern` defines United States Eastern Standard Time and Eastern Daylight Time. The time zone settings used by applications are read (by *daytime*(2)) from the file `/locale/timezone`, which is initially a copy of `/locale/GB-Eire`. If displaying time as the time in London is adequate, you need change nothing. To set a different time zone for the whole site, copy the appropriate time zone file into `/locale/timezone`:

```
cp /locale/US_Eastern /locale/timezone
```

To set a different time zone for a user or window, *bind*(1) the file containing the time zone setting over `/locale/timezone`, either in the user's profile or in a name space description file:

```
bind /locale/US_Eastern /locale/timezone
```

5. Running the Window Manager *wm*

Graphical Inferno programs normally run under the window manager *wm*(1). Inferno has a simple editor, *wm/edit*, that can be used to edit the inferno configuration files. The 'power environment' for editing and program development is *acme*(1), but rather that throwing you in at the deep end, we shall stick to the simpler one for now. If you already know Acme from Plan 9, however, or perhaps Wily from Unix, feel free to use Inferno's *acme* instead of *edit*.

- **Step 5a: Start the window manager.**

Invoke *wm* by typing

```
wm/wm
```

You should see a new window open with a blue-grey background and a small *Vita Nuova* logo in the bottom left hand corner. Click on the logo with mouse button 1 to reveal a small menu. Selecting the *Edit* entry will start *wm/edit*. In common with most *wm* programs the editor has three small buttons in a line at its top right hand corner. Clicking on the X button, the rightmost button, will close the program down. The leftmost of the three buttons will allow the window to be resized – after clicking it drag the window from a point near to either one of its edges or one of its corners. The middle button will minimise the window, creating an entry for it in the application bar along the bottom of the main *wm* window. You can restore a minimised window by clicking on its entry in the application bar. The initial *wm* configuration is determined by the contents of the shell script `/lib/wmsetup` (see *toolbar*(1) and *sh*(1)).

- **Step 5b: Open a shell window.**

Choose the *shell* option from the menu to open up a shell window. The configuration of Inferno will be done from this shell window.

6. Manual Pages

Manual pages for all of the system commands are available from a shell window. Use the *man* or *wm/man* commands. For example,

```
man wm
```

will give information about *wm*. And

```
man man
```

will give information about using *man*. *Wm/man* makes use of the Tk text widget to produce slightly more attractive output than the plain command *man*. Here, and in other Inferno documentation you will see references to manual page entries of the form *command*`(`*section*`)`. You can display the manual page for the command by running

`man` *command*

or

`man` *section command*

if the manual page appears in more than one section.

7. Initial Namespace

The initial Inferno namespace is built by placing the root device '`#/`' (see *root*(3)) at the root of the namespace and binding

i) the host filesystem device '#U' (see *fs*(3)) containing the *inferno_root* subtree of the host filesystem at the root of the Inferno filesystem,

ii) the console device '#c' (see *cons*(3)) in `/dev`,

iii) the prog device '#p' (see *prog*(3)) onto `/prog`,

iv) the IP device '#I' (see *ip*(3)) in `/net`, and

v) the environment device '#e' (see *env*(3)) at `/dev/env`.

You can see the sequence of commands required to construct the current namespace by running

```
ns
```

8. Inferno's network

If you are just going to use Inferno for local Limbo programming, and not use its networking interface, you can skip to the section ''Adding new users'' at the end of this document. You can always come back to this step later.

To use IP networking, the IP device *ip*(3)) must have been bound into `/net`. Typing

```
ls -l /net
```

(see *ls*(1)) should result in something like

```
--rw-rw-r-- I 0 network john 0 May 31 07:11 /net/arp
--rw-rw-r-- I 0 network john 0 May 31 07:11 /net/ndb
d-r-xr-xr-x I 0 network john 0 May 31 07:11 /net/tcp
d-r-xr-xr-x I 0 network john 0 May 31 07:11 /net/udp
```

There might be many more names on some systems.

A system running Inferno, whether native or hosted, can by agreement attach to any or all resources that are in the name space of another Inferno system (or even its own). That requires:

- the importing system must know where to find them
- the exporting system must agree to export them

- the two systems must authenticate the access (not all resources will be permitted to all systems or users)
- the conversation can be encrypted to keep it safe from prying eyes and interference

On an Inferno network, there is usually one secure machine that acts as authentication server. All other systems variously play the rôles of server and client as required: any system can import some resources (or none) and export others (or none), simultaneously, and differently in different name spaces. In following sections, we shall write as though there were three distinct machines: authentication server (signer); server (exporting resources); and client (importing resources). With Inferno, you can achieve a similar effect on a single machine by starting up distinct instances of *emu* instead. That is the easiest way to become familiar with the process (and also avoids having to install the system on several machines to start). It is still worthwhile setting up a secured authentication server later, especially if you are using Windows on a FAT file system where the host system's file protections are limited.

We shall now configure Inferno to allow each of the functions listed above:

- change the network database to tell where to find local network resources
- set up the authentication system, specifically the authentication server or 'signer'
- start network services (two distinct sets: one for the authentication services and the other for all other network services)

9. Network database files

In both hosted and native modes, Inferno uses a collection of text files of a particular form to store all details of network and service configuration. When running hosted, Inferno typically gets most of its data from the host operating system, and the database contains mainly Inferno-specific data.

The file `/lib/ndb/local` is the root of the collection of network database files. The format is defined by *ndb*(6), but essentially it is a collection of groups of attribute/value pairs of the form *attribute*=*value*. Attribute names and most values are case-sensitive.

Related attribute/value pairs are grouped into database 'entries'. An entry can span one or more lines: the first line starts with a non-blank character, and any subsequent lines in that entry start with white space (blank or tab).

9.1. Site parameters

The version of `/lib/ndb/local` at time of writing looks like this:

```
database=
    file=/lib/ndb/local
    file=/lib/ndb/dns
    file=/lib/ndb/inferno
    file=/lib/ndb/common

infernosite=
    #dnsdomain=your.domain.com
    #dns=1.2.3.4    # resolver
    SIGNER=your_Inferno_signer_here
    FILESERVER=your_Inferno_fileserver_here
    smtp=your_smtpserver_here
    pop3=your_pop3server_here
    registry=your_registry_server
```

The individual files forming the data base are listed in order in the `database` entry. They can be ignored for the moment. The entry labelled `infernosite=` defines a mapping from symbolic host names of the form $*service* to a host name, domain name, or a numeric Internet address. For instance, an application that needs an authentication service will refer to `$SIGNER` and an Inferno naming service will translate that at run-time to the appropriate network name for that environment. Consequently, the entries above need to be customised for a given site. (The items that are commented out are not needed when the host's own DNS resolver is used instead of Inferno's own *dns*(8).) For example, our `infernosite` entry in the `local` file might look something like this

```
infernosite=
    dnsdomain=vitanuova.com
    dns=200.1.1.11  # resolver
    SIGNER=doppio
    FILESERVER=doppio
    smtp=doppio
    pop3=doppio
    registry=doppio
```

where `doppio` is the host name of a machine that is offering the given services to Inferno, and `200.1.1.11` is the Internet address of a local DNS resolver.

- **Step 9a: Enter defaults for your site**

The only important names initially are:

`SIGNER`	the host or domain name, or address of the machine that will act as signer
`registry`	the name or address of a machine that provides the local dynamic service *registry*(4)
`FILESERVER`	the primary file server (actually needed only by clients with no storage of their own)

All others are used by specific applications such as *acme*(1) mail or *ftpfs*(4).

If you are using a single machine for signer and server/client, put its name in those three entries.

9.2. Connection server *cs*(8) and name translation

The connection server *cs*(8) uses the network database and other data (such as that provided by the host system and the Internet DNS servers) to translate symbolic network names and services into instructions for connecting to a given service. In hosted mode, network and service names are passed through to the host for conversion to numeric IP addresses and port numbers. If the host is unable to convert a service name the connection server will attempt to convert the name using mappings of service and protocol names to Internet port numbers in the file `/lib/ndb/inferno`:

```
tcp=infgamelogin port=6660  # inferno games login service
tcp=styx port=6666  # main file service
tcp=mpeg port=6667  # mpeg stream
tcp=rstyx port=6668 # remote invocation
tcp=infdb port=6669 # database server
tcp=infweb port=6670    # inferno web server
tcp=infsigner port=6671 # inferno signing services
tcp=infcsigner port=6672    # inferno countersigner
tcp=inflogin port=6673  # inferno credential service
tcp=infsds port=6674    # software download
tcp=registry port=6675  # registry(4)
udp=virgil port=2202    # naming service
```

For the moment, leave that file as it is. You will only need to modify this file if in future you add new statically-configured services to Inferno. (Services that come and go dynamically might use *registry*(4) instead, a registry manager that allows a service to be found using a description of it.)

10. Configuring a Signer

Before an Inferno machine can authenticate establish a secure connection to an Inferno service on another machine, each system needs to obtain a certificate from a common signer.† We talk here as though there is only one 'signer' per site but in fact there can be application- or group-specific ones. For instance, a version of the Inferno games server automatically starts its own signing service to keep the identities and keys used for game service separate from those of the primary system, allowing users to set up their own gaming groups without fuss. To use authenticated connections for the primary file services we need to set up a signer to generate certificates for users (see *createsignerkey*(8) and *logind*(8)).

Choose an Inferno machine to become the signer. If this is the first or only Inferno machine on your network then make this machine the signer. (It is more realistic if you start up a separate copy of *emu* and

†The authentication system will shortly expand to a rôle-based one allowing a chain of certificates to be used, from several signers, with delegation etc.

leave it in 'console' mode without starting the window system.) You can always move the function elsewhere later.

- **Step 10a: Empty the secret file of secrets.**

The authentication server verifies a user's identity by checking that the user knows a shared secret. (In fact the secret is not used directly, but instead a scrambled value that was derived from it.) The file `/keydb/keys` holds those secrets; it is encrypted using a secret password or phrase known only to the manager of the authentication server. Having just installed Inferno, the file should exist and be readable only by you (or the user as which the authentication service will run). On the signer machine, type

```
ls -l /keydb/keys
```

You should see something like:

```
--rw------- M 7772 yourname inf 0 Jun 12 03:08 /keydb/keys
```

You should see something like the above. If the file does not exist or is not empty or has the wrong mode, use:

```
cp /dev/null /keydb/keys; chmod 600 /keydb/keys
```

to set it right.

- **Step 10b: Generate a signer key.**

Next on the signer machine, run

```
auth/createsignerkey name
```

In place of *name* enter the network name of the signer. A fully-qualified domain name of a host or individual is fine. This value will appear as the signer name in each certificate generated by the signer. *Createsignerkey* creates public and private keys that are used by the signer when generating certificates. They are stored in `/keydb/signerkey`; check that it has permissions that limit access to the user that will run the authentication services:

```
; ls -l /keydb/signerkey
--rw------- M 32685 secrets inf 1010 Jul 07  2000 /keydb/signerkey
```

Use *chmod*(1) to set the mode to read/write only for the owner if necessary:

```
chmod 600 /keydb/signerkey
```

- **Step 10c: Start the authentication network services**

Still at the signer console, type

```
svc/auth
```

That script (see *svc*(8)) will check that the `/keydb/keys` and `/keydb/signerkey` files exist, and then start the program *keyfs*(4), which manages the `keys` file. It will prompt for the password (pass phrase) you wish to use to protect the `keys` file, now and on subsequent runs:

```
; svc/auth
Key:
Confirm key:
```

It prompts twice to confirm it. If successfully confirmed, it will then start the network services used by Inferno to authenticate local and remote users and hosts. (If confirmation fails, retry by running `svc/auth` again.)

You can check that they are running by typing:

```
ps
```

which should show something like the following:

```
 1        1    john    release    74K Sh[$Sys]
 3        2    john        alt    15K Cs
10        9    john       recv    25K Keyfs
11        9    john    release    44K Styx[$Sys]
12        9    john       recv    25K Keyfs
14        1    john        alt     8K Listen
16        1    john    release     8K Listen[$Sys]
18        1    john        alt     9K Listen
20        1    john    release     9K Listen[$Sys]
22        1    john        alt     9K Listen
24        1    john    release     9K Listen[$Sys]
26        1    john        alt     8K Listen
28        1    john    release     8K Listen[$Sys]
29        1    john      ready    73K Ps[$Sys]
```

There should be `Keyfs` and `Listen` processes.

- **Step 10d: Enter user names and secrets.**

For each user to be authenticated by the signer run

```
auth/changelogin username
```

You will be prompted to supply a secret (ie, password or pass phrase) and expiration date. The expiration date will be used as the maximum expiration date of authentication certificates granted to that user. *Changelogin* (see *changelogin*(8)) accesses the name space generated by *keyfs*(4), which has just been started above by `svc/auth`. A user can later change the stored secret using the *passwd*(1) command. For the signer to generate a certificate there must be at least one entry in the password file. If you are not sure at this stage of the names of the users that you want to authenticate then create an entry for the user `inferno` and yourself.

11. Establishing a Secure Connection

To establish a secure connection between two Inferno machines, each needs to present a public key with a certificate signed by a common signer. We shall make two public/private key sets, one for the server and one for a client (they differ only in where they are stored). We shall do the server first, because the usual network services require the server possess some keys before they can start. We shall then start those services, and finally sort out the client.

- **Step 11a: Start the connection service.**

The server still needs to make contact with the signer, so we need to start the basic connection service *cs*(8). If you are using the same instance of *emu* in which you invoked `svc/auth` above, you should skip this step. To check, you should see a new file in the `/net` directory called `cs`. Run the command

```
ls /net
```

You should see at least the following names in the output:

```
/net/cs
/net/ndb
/net/tcp
/net/udp
```

Otherwise, type

```
ndb/cs
```

That starts *cs*(8). Try the `ls /net` again to check that the `cs` file has appeared.

- **Step 11b: Generate a server key set.**

On the server machine (or in the 'server' window), use *getauthinfo*(8) to obtain a certificate and save it in a file named `default` by running

```
getauthinfo default
```

Getauthinfo will prompt for the address of your signer (you can often just use its host name or even `localhost`) and for a remote username and password combination. *Getauthinfo* will connect to the *inflogin*

service on the signer and authenticate you against its user and password database, `/keydb/keys`, using the username and password that you entered above. Answer `yes` to the question that asks if you want to save the certificate in a file. *Getauthinfo* will save a certificate in the file `/usr/`*user*`/keyring/default` where *user* is the name in `/dev/user`.

12. Network Services

As mentioned above, in a full Inferno network the authentication services will usually be run on a secured machine of their own (the signer), and the ordinary network services such as file service are not run on a signer. If you are, however, using one machine for all functions, you can get the right effect by starting another instance of *emu*, to act as an Inferno host that is not a signer. This one will run the services of a primary file server and the site *registry*(4).

Commands described in *svc*(8) start listeners for various local network services. (The commands are actually shell scripts.) As we saw above, `svc/auth` starts the services on a signer.

Here we shall start the usual set of services.

- **Step 12a: Start the network listener services.**

Type

```
svc/net
```

Various network services will (should!) now be running. To confirm this type

```
ps
```

which should show something like the following:

```
; ps
       1        1    inferno    release    74K Sh[$Sys]
       7        6    inferno        alt    15K Cs
      13        1    inferno       recv    15K Registry
      14        1    inferno    release    44K Styx[$Sys]
      15        1    inferno       recv    15K Registry
      17        1    inferno        alt     8K Listen
      19        1    inferno    release     8K Listen[$Sys]
      22        1    inferno        alt     8K Listen
      24        1    inferno    release     8K Listen[$Sys]
      25        1    inferno      ready    74K Ps[$Sys]
```

There should be a few `Listen` processes and perhaps a `Registry`.

You can also try

```
netstat
```

Netstat prints information about network connections. You should see several lines of output, each one describing an announced TCP or UDP service. Depending upon the contents of the network configuration files we included on the CD, you might see output something like this:

```
tcp/1      Announced    inferno    200.1.1.89!6668      ::!0
tcp/2      Announced    inferno    200.1.1.89!6666      ::!0
tcp/3      Announced    inferno    200.1.1.89!6675      ::!0
```

Each line corresponds to a network connection: the connection name, the name of the user running the server, the address of the local end of the connection, the address of the remote end of the connection, and the connection status. The connection name is actually the protocol and conversation directory in `/net`. The connection addresses are all of the form *host*`!`*port* for these IP based services, and the remote addresses are not filled in because they all represent listening services that are in the `Announced` state. In this example the third line shows a TCP service listening on port 6675. Examining `/lib/ndb/inferno` with `grep` (see *grep*(1)) shows that the listener on port 6675 is the Inferno registry service

```
grep 6675 /lib/ndb/inferno
```

gives

```
tcp=registry port=6675  # default registry
```

Now the server is ready but we need a client.

Either use a third machine or (more likely at first) simply start another *emu* instance in a new window. Start its connection server, again by typing

```
ndb/cs
```

The connection server is fundamental to the Inferno network. Once networking is set up, when subsequently starting up a client you should start *cs* before starting the window system. Note that if you are running the Inferno instance as a server, or combined server and client, the `svc/net` that starts the network services automatically starts *cs*, and you need not do so explicitly.

- **Step 12b: Generate a client certificate.**

Obtain a certificate for the client in the same way as on the server. We shall obtain a certificate for use with a specific server by storing it in a file whose name exactly matches the network address of the server

```
getauthinfo tcp!hostname
```

Use the current machine's *hostname*. *Getauthinfo* stores the certificate in the file `/usr/`*user*`/keyring/`*keyname* where *user* is the name in `/dev/user` and *keyname* is the argument given to *getauthinfo*. Again, answer `yes` to the question that asks if you want to save the certificate in a file.

Now that both client and server have a certificate obtained from the same signer it is possible to establish a secure connection between them. Note that getting keys and certificates with *getauthinfo* is normally done just once (or at most once per server when the `default` key is not used). In short, all the work done up to now need not be repeated. After this, provided the keys were saved to a keyring file, as many authenticated connections can be made as desired until the certificate expires (which by default is whenever the password entry was set to expire). Also note that the certificates for different machines can have different signers, and one can even use different certificates for the same machine when the remote user name is to differ (the `-f` option of *getauthinfo* can then be useful to force an appropriate keyring name).

- **Step 12c: Make an authenticated connection.**

The script `svc/net` on the server started fundamental name services and also a Styx file service. That can also be started separately using `svc/styx`. In either case the namespace that is served is the one in which the command was invoked. Now you can test the service.

Confirm that `/n/remote` is an empty directory by typing

```
lc /n/remote
```

You can now mount the server's name space onto the client's directory `/n/remote` by typing

```
mount  serveraddr /n/remote
```

Where *serveraddr* is the IP address of the server or a name that the host can resolve to the IP address of the server. Now

```
lc /n/remote
```

should reveal the files and directories in the namespace being served by the server. Those files are now also visible in the namespace of your shell. You will notice that these changes only affect the shell in which you ran the *mount* command – other windows are unaffected. You can create, remove or modify files and directories in and under `/n/remote` much as you can any other file or directory in your namespace. In fact, in general, a process does not need to know whether a file actually resides locally or remotely. You can unmount the mounted directory with *unmount*. Type

```
unmount /n/remote
```

You can confirm that it has gone by running

```
ls /n/remote
```

All connections made by Inferno are authenticated. The default connection made by *mount* is authenticated but uses neither encryption nor secure digests. You can pass an argument to *mount* to specify a more secure connection: its `-C` option gives it a hashing and an encryption algorithm to be applied to the connection.

- **Step 12d: Make a secure authenticated connection.**

For example,

```
mount  -C sha1/rc4_256 serveraddr /n/remote
```

makes an authenticated connection to the machine given by *serveraddr*, then engages SHA1 hashing for message digesting and 256-bit RC4 for encryption.

It mounts the namespace served by *serveraddr*'s Styx service on the local Inferno directory `/n/remote`.

13. Adding new users

Every inferno process has an associated *user name*. At boot time the user name is set equal to your login name on the host operating system. The user name is used by *wm/logon* to select the home directory, and by other programs like *mount* when it searches for certificates. (It can also control permission for file access on the local system in native Inferno and some hosted Inferno configurations.) When you attach to a server on another system the user name in the authenticating certificate can be used by the remote file service to set the user name appropriately there.†

To create a new user, copy the directory `/usr/inferno` into `/usr/`*username*. If the user is to have access to services on the network, make an authentication server entry using *changelogin*(8). The user can change the stored secret using *passwd*(1), if desired. Having logged in for the first time, the user should generate a default public/private key set using *getauthinfo*(8). (The authentication services must be running somewhere.)

The *wm* window manager program *wm/logon* allows a user to login to the local Inferno system as an Inferno user (different from the host user name). Its use is shown next.

- **Step 13a: Re-start Inferno.**

You should now close down any instances of *emu* that you are currently running. The quickest way to do this is to type *control-c* in the emu window in which you ran *wm/wm*. Start a new *emu*, as before, by either running

```
emu
```

or by choosing the appropriate entry from your start menu on Windows machines. This time, start network services

```
svc/net
```

and then run

```
wm/wm wm/logon
```

and log in as user *inferno*. When you log in, *wm/logon* will change directory to `/usr/inferno` and then write the name `inferno` to `/dev/user`. If the file `/usr/inferno/namespace` exists it will be used to construct a new namespace for the user based on the commands that it contains (see *newns*(2)).

14. What next

You should now have a fully functional Inferno system. You will need to have a three button mouse to use *acme*, *wm*, or *plumbing*.

To learn more you could start with the manual pages for: *intro*(1), *emu*(1), *wm*(1), *wm-misc*(1), *sh*(1), *acme*(1), and *limbo*(1) and also the papers in sections 1, 2 and 3 of Volume 2 of *The Inferno Programmer's Manual*.

†The details are system-dependent and currently subject to change.