UNIX Administration Course

Version 1.0

mapleson@gamers.org
Tel: (+44) (0)1772 893297
Fax: (+44) (0)1772 892913
WWW: http://www.futuretech.vuurwerk.nl/

Detailed Notes for Day 2 (Part 1)

UNIX Fundamentals: System Identity, IP Address, Domain Name, Subdomain.

Every UNIX system has its own unique name, which is the means by which that machine is referenced on local networks and beyond, eg. the Internet. The normal term for this name is the local 'host' name. Systems connected to the Internet employ naming structures that conform to existing structures already used on the Internet. A completely isolated network can use any naming scheme.

Under IRIX, the host name for a system is stored in the /etc/sys_id file. The name may be up to 64 alphanumeric characters in length and can include hyphens and periods. Period characters '.' are not part of the real name but instead are used to separate the sequence into a domain-name style structure (eg. www.futuretech.vuurwerk.nl). The SGI server's host name is yoda, the fully-qualified version of which is written as yoda.comp.uclan.ac.uk. The choice of host names is largely arbitrary, eg. the SGI network host names are drawn from my video library (I have chosen names designed to be short without being too uninteresting).

On bootup, a system's /etc/rc2.d/S20sysetup script reads its /etc/sys_id file to determine the local host name. From then onwards, various system commands and internal function calls will return that system name, eg. the 'hostname' and 'uname' commands (see the respective man pages for details).

Along with a unique identity in the form of a host name, a UNIX system has its own 32bit Internet Protocol (IP) address, split for convenience into four 8bit integers separated by periods, eg. yoda's IP address is 193.61.250.34, an address which is visible to any system anywhere on the Internet.

IP is the network-level communications protocol used by Internet systems and services. Various extra options can be used with IP layer communications to create higher-level services such as TCP (Transmission Control Protocol). The entire Internet uses the TCP/IP protocols for communication.

A system which has more than one network interface (eg. multiple Ethernet ports) must have a unique IP address for each port. Special software may permit a system to have extra addresses, eg. 'IP Aliasing', a technique often used by an ISP to provide a more flexible service to its customers. Note: unlike predefined Ethernet addresses (every Ethernet card has its own unique address), a system's IP address is determined by the network design, admin personnel, and external authorities.

Conceptually speaking, an IP address consists of two numbers: one represents the network while the other represents the system. In order to more efficiently make use of the numerous possible address 'spaces', four classes of addresses exist, named A, B, C and D. The first few bits of an address determines its class:

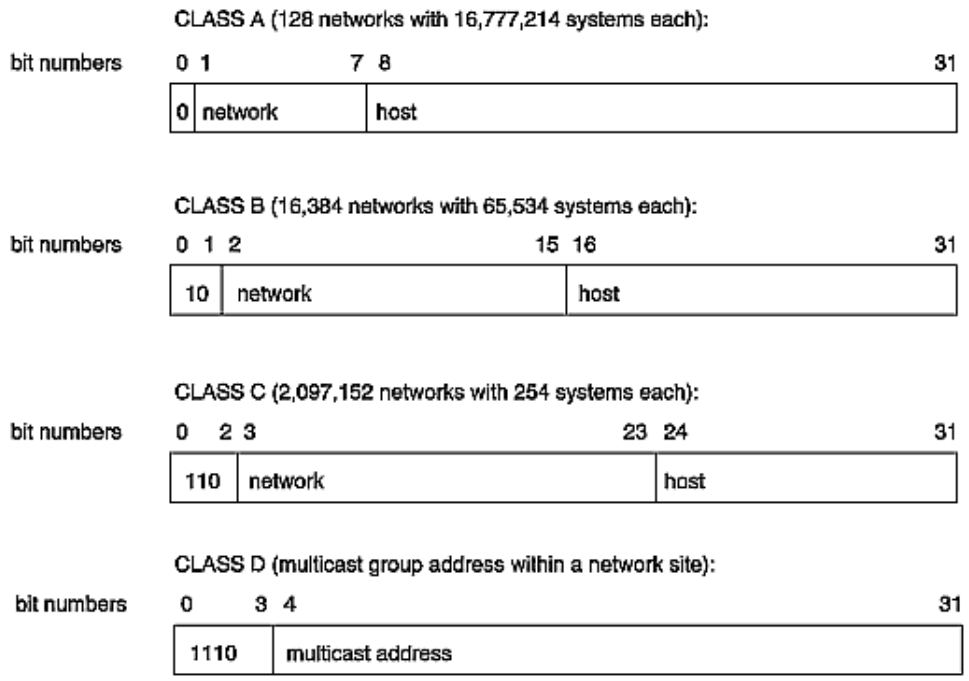
Class	Initial Binary Bit Field	No. of Bits for the Network Number	No. of Bits for The Host Number
A	0	7	24
B	10	14	16
C	110	21	8
D	1110	[special 'multicast' addresses for internal network use]	

Figure 20. IP Address Classes: bit field and width allocations.

This system allows the Internet to support a range of different network sizes with differing maximum limits on the number of systems for each type of network:

	Class A	Class B	Class C	Class D
No. of networks:	128	16384	2097152	[multicast]
No. of systems each:	16777214	65534	254	[multicast]

Figure 21. IP Address Classes: supported network types and sizes.



The numbers 0 and 255 are never used for any host. These are reserved for special uses.

Note that a network which will never be connected to the Internet can theoretically use any IP address and domain/subdomain configuration.

Which class of network an organisation uses depends on how many systems it expects to have within its network. Organisations are allocated IP address spaces by Internet Network Information Centers (InterNICs), or by their local ISP if that is how they are connected to the Internet. An organisation's domain name (eg. uclan.ac.uk) is also obtained from the local InterNIC or ISP. Once a domain name has been allocated, the organisation is free to setup its own network subdomains such as comp.uclan.ac.uk (comp = Computing Department), within which an individual host would be yoda.comp.uclan.ac.uk. A similar example is Heriot Watt University in Edinburgh (where I

studied for my BSc) which has the domain hw.ac.uk, with its Department of Computer Science and Electrical Engineering using a subdomain called cee.hw.ac.uk, such that a particular host is www.cee.hw.ac.uk (see Appendix A for an example of what happens when this methodology is not followed correctly).

UCLAN uses Class C addresses, with example address spaces being 193.61.255 and 193.61.250. A small number of machines in the Computing Department use the 250 address space, namely the SGI server's external Ethernet port at 193.61.250.34, and the NT server at 193.61.250.35 which serves the NT network in Ve27.

Yoda has two Ethernet ports; the remaining port is used to connect to the SGI Indys via a hub - this port has been defined to use a different address space, namely 193.61.252. The machines' IP addresses range from 193.61.252.1 for yoda, to 193.61.252.23 for the admin Indy; .20 to .22 are kept available for two HP systems which are occasionally connected to the network, and for a future plan to include Apple Macs on the network.

The IP addresses of the Indys using the 252 address space cannot be directly accessed outside the SGI network or, as the jargon goes, 'on the other side' of the server's Ethernet port which is being used for the internal network. This automatically imposes a degree of security at the physical level.

IP addresses and host names for systems on the local network are brought together in the file /etc/hosts. Each line in this file gives an IP address, an official hostname and then any name aliases which represent the same system, eg. yoda.comp.uclan.ac.uk is also known as www.comp.uclan.ac.uk, or just yoda, or www, etc. When a system is first booted, the ifconfig command uses the /etc/hosts file to assign addresses to the various available Ethernet network interfaces. Enter 'more /etc/hosts' or 'nedit /etc/hosts' to examine the host names file for the particular system you're using.

NB: due to the Internet's incredible expansion in recent years, the world is actually beginning to run out of available IP addresses and domain names; at best, existing top-level domains are being heavily overused (eg. .com, .org, etc.) and the number of allocatable network address spaces is rapidly diminishing, especially if one considers the possible expansion of the Internet into Russia, China, the Far East, Middle East, Africa, Asia and Latin America. Thus, there are moves afoot to change the Internet so that it uses 128bit instead of 32bit IP addresses. When this will happen is unknown, but such a change would solve the problem.

Special IP Addresses

Certain reserved IP addresses have special meanings, eg. the address 127.0.0.1 is known as the 'loopback' address (equivalent host name 'localhost') and always refers to the local system which one happens to be using at the time. If one never intends to connect a system to the Internet, there's no reason why this default IP address can't be left as it is with whatever default name assigned to it in the /etc/hosts file (SGIs always use the default name, "IRIS"), though most people do change their system's IP address and host name in case, for example, they have to connect their system to the network used at their place of work, or to provide a common naming scheme, group ID setup, etc.

If a system's IP address is changed from the default 127.0.0.1, the exact procedure is to add a new line to the /etc/hosts file such that the system name corresponds to the information in /etc/sys_id. One must *never* remove the 127.0.0.1 entry from the /etc/hosts file or the system will not work

properly. The important lines of the /etc/hosts file used on the SGI network are shown in Fig 22 below (the appearance of '[etc]' in Fig 22 means some text has been clipped away to aid clarity).

```
# This entry must be present or the system will not work.
127.0.0.1       localhost

# SGI Server. Challenge S.
193.61.252.1    yoda.comp.uclan.ac.uk yoda www.comp.uclan.ac.uk www [etc]

# Computing Services router box link.
193.61.250.34   gate-yoda.comp.uclan.ac.uk gate-yoda

# SGI Indys in Ve24, except milamber which is in Ve47.

193.61.252.2    akira.comp.uclan.ac.uk akira
193.61.252.3    ash.comp.uclan.ac.uk ash
193.61.252.4    cameron.comp.uclan.ac.uk cameron
193.61.252.5    chan.comp.uclan.ac.uk chan
193.61.252.6    conan.comp.uclan.ac.uk conan
193.61.252.7    gibson.comp.uclan.ac.uk gibson
193.61.252.8    indiana.comp.uclan.ac.uk indiana
193.61.252.9    leon.comp.uclan.ac.uk leon
193.61.252.10   merlin.comp.uclan.ac.uk merlin
193.61.252.11   nikita.comp.uclan.ac.uk nikita
193.61.252.12   ridley.comp.uclan.ac.uk ridley
193.61.252.13   sevrin.comp.uclan.ac.uk sevrin
193.61.252.14   solo.comp.uclan.ac.uk solo
193.61.252.15   spock.comp.uclan.ac.uk spock
193.61.252.16   stanley.comp.uclan.ac.uk stanley
193.61.252.17   warlock.comp.uclan.ac.uk warlock
193.61.252.18   wolfen.comp.uclan.ac.uk wolfen
193.61.252.19   woo.comp.uclan.ac.uk woo
193.61.252.23   milamber.comp.uclan.ac.uk milamber

[etc]
```

Figure 22. The contents of the /etc/hosts file used on the SGI network.

One example use of the localhost address is when a user accesses a system's local web page structure at:

```
http://localhost/
```

On SGIs, such an address brings up a page about the machine the user is using. For the SGI network, the above URL always brings up a page for yoda since /var/www is NFS-mounted from yoda. The concept of a local web page structure for each machine is more relevant in company Intranet environments where each employee probably has her or his own machine, or where different machines have different locally stored web page information structures due to, for example, differences in available applications, etc.

The BIND Name Server (DNS).

If a site is to be connected to the Internet, then it should use a name server such as BIND (Berkeley Internet Name Domain) to provide an Internet Domain Names Service (DNS). DNS is an Internet-standard name service for translating hostnames into IP addresses and vice-versa. A client machine wishing to access a remote host executes a query which is answered by the DNS daemon, called 'named'. Yoda runs a DNS server and also a Proxy server, allowing the machines in Ve24 to access the Internet via Netscape (telnet, ftp, http, gopher and other services can be used).

Most of the relevant database configuration files for a DNS setup reside in /var/named. A set of example configuration files are provided in /var/named/Examples - these should be used as templates and modified to reflect the desired configuration. Setting up a DNS database can be a little confusing at first, thus the provision of the Examples directory. The files which must be

configured to provide a functional DNS are:

```
/etc/named.boot
/var/named/root.cache
/var/named/named.hosts
/var/named/named.rev
/var/named/localhost.rev
```

If an admin wishes to use a configuration file other than /etc/named.boot, then its location should be specified by creating a file called /etc/config/named.options with the following contents (or added to named.options if it already exists):

```
-b some-other-boot-file
```

After the files in /var/named have been correctly configured, the chkconfig command is used to set the appropriate variable file in /etc/config:

```
chkconfig named on
```

The next reboot will activate the DNS service. Once started, named reads initial configuration information from the file /etc/named.boot, such as what kind of server it should be, where the DNS database files are located, etc. Yoda's named.boot file looks like this:

```
;
; Named boot file for yoda.comp.uclan.ac.uk.
;
directory       /var/named

cache           .                               root.cache
primary         comp.uclan.ac.uk                named.hosts
primary         0.0.127.IN-ADDR.ARPA            localhost.rev
primary         252.61.193.IN-ADDR.ARPA         named.rev
primary         250.61.193.IN-ADDR.ARPA         250.rev
forwarders      193.61.255.3 193.61.255.4
```

Figure 23. Yoda's /etc/named.boot file.

Looking at the contents of the example named.boot file in /var/named/Examples, the differences are not that great:

```
;
; boot file for authoritative master name server for Berkeley.EDU
; Note that there should be one primary entry for each SOA record.
;
;
sortlist 10.0.0.0

directory       /var/named

; type    domain                source host/file                backup file

cache     .                     root.cache
primary   Berkeley.EDU          named.hosts
primary   32.128.IN-ADDR.ARPA   named.rev
primary   0.0.127.IN-ADDR.ARPA  localhost.rev
```

Figure 24. The example named.boot file in /var/named/Examples.

Yoda's file has an extra line for the /var/named/250.rev file; this was an experimental attempt to make Yoda's subdomain accessible outside UCLAN, which failed because of the particular configuration of a router box elsewhere in the communications chain (the intention was to enable students and staff to access the SGI network using telnet from a remote host).

For full details on how to configure a typical DNS, see Chapter 6 of the online book, "IRIX Admin: Networking and Mail". A copy of this Chapter has been provided for reference. As an example of how identical DNS is across UNIX systems, see the issue of Network Week [10] which has an article on configuring a typical DNS. Also, a copy of each of Yoda's DNS files which I had to configure are included for reference. Together, these references should serve as an adequate guide to configuring a DNS; as with many aspects of managing a UNIX system, learning how someone else solved a problem and then modifying copies of what they did can be very effective.

Note: it is not always wise to use a GUI tool for configuring a service such as BIND [11]. It's too easy for ill-tested grandiose software management tools to make poor assumptions about how an admin wishes to configure a service/network/system. Services such as BIND come with their own example configuration files anyway; following these files as a guide may be considerably easier than using a GUI tool which itself can cause problems created by whoever wrote the GUI tool, rather than the service itself (in this case BIND).

Proxy Servers

A Proxy server acts as a go-between to the outside world, answering client requests for data from the Internet, calling the DNS system to obtain IP addresses based on domain names, opening connections to the Internet perhaps via yet another Proxy server elsewhere (the Ve24 system uses Pipex as the next link in the communications chain), and retrieving data from remote hosts for transmission back to clients.

Proxy servers are a useful way of providing Internet access to client systems at the same time as imposing a level of security against the outside world, ie. the internal structure of a network is hidden from the outside world due to the operational methods employed by a Proxy server, rather like the way in which a representative at an auction can act for an anonymous client via a mobile phone during the bidding. Although there are more than a dozen systems in Ve24, no matter which machine a user decides to access the Internet from, the access will always appear to a remote host to be coming from the IP address of the closest proxy server, eg. the University web server would see Yoda as the accessing client. Similarly, I have noticed that when I access my own web site in Holland, the site concerned sees my access as if it had come from the proxy server at Pipex, ie. the Dutch system cannot see 'past' the Pipex Proxy server.

There are various proxy server software solutions available. A typical package which is easy to install and configure is the Netscape Proxy Server. Yoda uses this particular system.

Network Information Service (NIS)

It is reasonably easy to ensure that all systems on a small network have consistent /etc/hosts files using commands such as rcp. However, medium-sized networks consisting of dozens to hundreds of machines may present problems for administrators, especially if the overall setup consists of several distinct networks, perhaps in different buildings and run by different people. For such environments, a Network Information Service (NIS) can be useful. NIS uses a single system on the network to act as the sole trusted source of name service information - this system is known as the NIS master. Slave servers may be used to which copies of the database on the NIS master are periodically sent, providing backup services should the NIS master system fail.

Client systems locate a name server when required, requesting data based on a domain name and

other relevant information.

Unified Name Service Daemon (UNS, or more commonly NSD).

Extremely recently, the DNS and NIS systems have been superseded by a new system called the Unified Name Service Daemon, or NSD for short. NSD handles requests for domain information in a considerably more efficient manner, involving fewer system calls, replacing multiple files for older services with a single file (eg. many of the DNS files in /var/named are replaced by a single database file under NSD), and allowing for much larger numbers of entries in data files, etc.

However, NSD is so new that even I have not yet had an opportunity to examine properly how it works, or the way in which it correlates to the older DNS and NIS services. As a result, this course does not describe DNS, NIS or NSD in any great detail. This is because, given the rapid advance of modern UNIX OSs, explaining the workings of DNS or NIS would likely be a pointless task since any admin beginning her or his career now is more likely to encounter the newer NSD system which I am not yet comfortable with. Nevertheless, administrators should be aware of the older style services as they may have to deal with them, especially on legacy systems. Thus, though not discussed in these lectures, some notes on a typical DNS setup are provided for further reading [10]. Feel free to login to the SGI server yourself with:

```
rlogin yoda
```

and examine the DNS and NIS configuration files at your leisure; these may be found in the /var/named and /var/yp directories. Consult the online administration books for further details.

UNIX Fundamentals: UNIX Software Features

Software found on UNIX systems can be classified into several types:

- System software: items provided by the vendor as standard.
- Commercial software: items purchased either from the same vendor which supplied the OS, or from some other commercial 3rd-party.
- Shareware software: items either supplied with the OS, or downloaded from the Internet, or obtained from some other source such as a cover magazine CD.
- Freeware software: items supplied in the same manner as Shareware, but using a more open ’conditions of use’.
- User software: items created by users of a system, whether that user is an admin or an ordinary user.

System Software

Any OS for any system today is normally supplied on a set of CDs. As the amount of data for an OS installation increases, perhaps the day is not far away when vendors will begin using DVDs instead.

Whether or not an original copy of OS CDs can be installed on a system depends very much on the particular vendor, OS and system concerned. Any version of IRIX can be installed on an SGI system which supports that particular version of IRIX - this ability to install the OS whether or not one has a legal right to use the software is simply a practice SGI has adopted over the years. SGI could have chosen to make OS installation more difficult by requiring license codes and other details at installation time, but instead SGI chose a different route. What is described here applies only to SGI's IRIX OS.

SGI decided some time ago to adopt a strategy of official software and hardware management which makes it extremely difficult to make use of 'pirated' software. The means by which this is achieved is explained in the System Hardware section below, but the end result is a policy where any version IRIX older than the 'current' version is free by default. Thus, since the current release of IRIX is 6.5, one could install IRIX 6.4, 6.3, 6.2 (or any older version) on any appropriate SGI system (eg. installing IRIX 6.2 on a 2nd-hand Indy) without having to worry about legal issues. There's nothing to stop one physically installing 6.5 if one had the appropriate CDs (ie. the software installation tools and CDs do not include any form of installation protection or copy protection), but other factors might make for trouble later on if the user concerned did not apply for a license at a later date, eg. attempting to purchase commercial software and licenses for the latest OS release.

It is highly likely that in future years, UNIX vendors will also make their current OSs completely free, probably as a means of combating WindowsNT and other rivals.

As an educational site operating under an educational license agreement, UCLAN's Computing Department is entitled to install IRIX 6.5 on any of the SGI systems owned by the Computing Department, though at present most systems use the older IRIX 6.2 release for reasons connected with system resources on each machine (RAM, disk space, CPU power).

Thus, the idea of a license can have two meanings for SGIs:

- A theoretical 'legal' license requirement which applies, for example, to the current release of IRIX, namely IRIX 6.5 - this is a legal matter and doesn't physically affect the use of IRIX 6.5 OS CDs.

- A real license requirement for particular items of software using license codes, obtainable either from SGI or from whatever 3rd-party the software in question was purchased.

Another example of the first type is the GNU licensing system, explained in the 'Freeware Software' section below (what the GNU license is and how it works is fascinatingly unique).

Due to a very early top-down approach to managing system software, IRIX employs a high-level software installation structure which ensures that:

- It is extremely easy to add, remove, or update software, especially using the GUI software tool called Software Manager (swmgr is the text command name which can be entered in a shell).

- Changes to system software are handled correctly with very few, if any, errors most of the time; 'most' could be defined as 'rarely, if ever, but not never'. A real world example might be to state that I have installed SGI software elements thousands of times and rarely if ever encountered problems, though I have had to deal with some issues on occasion.

- Software 'patches' (modificational updates to existing software already installed) are handled in such a way as to allow the later removal of said patches if desired, leaving the system in exactly its original state as if the patch had never been installed.

As an example of software installation reliability, my own 2nd-hand Indigo2 at home has been in use since March 1998, was originally installed with IRIX 6.2, updated with patches several times, added to with extra software over the first few months of ownership (mid-1998), then upgraded to IRIX 6.5, added to with large amounts of freeware software, then upgraded to IRIX 6.5.1, then 6.5.2, then 6.5.3, and all without a single software installation error of any kind. In fact, my Indigo2 hasn't crashed or given a single error since I first purchased it. As is typical of any UNIX system which is/was widely used in various industries, most if not all of the problems ever encountered on the Indigo2 system have been resolved by now, producing an incredibly stable platform. In general, the newer the system and/or the newer the software, then the greater number of problems there will be to deal with, at least initially.

Thankfully, OS revisions largely build upon existing code and knowledge. Plus, since so many UNIX vendors have military, government and other important customers, there is incredible pressure to be very careful when planning changes to system or application software. Intensive testing is done *before* any new version is released into the marketplace (this contrasts completely with Microsoft which deliberately allows the public to test Beta versions of its OS revisions as a means of locating bugs before final release - a very lazy way to handle system testing by any measure).

Because patches often deal with release versions of software subsystems, and many software subsystems may have dependencies on other subsystems, the issue of patch installation is the most common area which can cause problems, usually due to unforseen conflicts between individual versions of specific files. However, rigorous testing and a top-down approach to tracking release versions minimises such problems, especially since all UNIX systems come supplied with source code version/revision tracking tools as-standard, eg. SCCS. The latest 'patch CD' can usually be installed automatically without causing any problems, though it is wise for an administrator to check what changes are going to be made before commencing any such installation, just in case.

The key to such a high-level software management system is the concept of a software 'subsystem'. SGI has developed a standard means by which a software suite and related files (manual pages, release notes, data, help documents, etc.) are packaged together in a form suitable for installation by the usual software installation tools such as inst and swmgr. Once this mechanism was carefully defined many years ago, insisting that all subsequent official software releases comply with the same standard ensures that the opportunity for error is greatly minimised, if not eliminated. Sometimes, certain 3rd-party applications such as Netscape can display apparent errors upon installation or update, but these errors are usually explained in accompanying documentation and can always be ignored.

Each software subsystem is usually split into several sub-units so that only relevant components need be installed as desired. The sub-units can then be examined to see the individual files which would be installed, and where. When making updates to software subsystems, selecting a newer version of a subsystem automatically selects only the relevant sub-units based on which sub-units have already been installed, ie. new items will not automatically be selected. For ease of use, an admin can always choose to execute an automatic installation or removal (as desired), though I often select a custom installation just so that I can see what's going on and learn more about the system as a result. In practice, I rarely need to alter the default behaviour anyway.

The software installation tools automatically take care not to overwrite existing configuration files when, for example, installing new versions (ie. upgrades) of software subsystems which have already been installed (eg. Netscape). In such cases, both the old and new configuration files are kept and the user (or admin) informed that there may be a need to decide which of the two files to keep, or perhaps to copy key data from the old file to the new file, deleting the old file afterwards.

Commercial Software

A 3rd-party commercial software package may or may not come supplied in a form which complies with any standards normally used by the hardware system vendor. UNIX has a long history of providing a generic means of packaging software and files in an archive which can be downloaded, uncompressed, dearchived, compiled and installed automatically, namely the 'tar.gz' archive format (see the man pages for tar and gzip). Many commercial software suppliers may decide to sell software in this format. This is ok, but it does mean one may not be able to use the usual software management tools (inst/swmgr in the case of SGIs) to later remove the software if desired. One would have to rely on the supplier being kind enough to either provide a script which can be used to remove the software, or at the very least a list of which files get installed where.

Thankfully, it is likely that most 3rd-parties will at least try to use the appropriate distribution format for a particular vendor's OS. However, unlike the source vendor, one cannot be sure that the 3rd-party has taken the same degree of care and attention to ensure they have used the distribution format correctly, eg. checking for conflicts with other software subsystems, providing product release notes, etc.

Commercial software for SGIs may or may not use the particular hardware feature of SGIs which SGI uses to prevent piracy, perhaps because exactly how it works is probably itself a licensed product from SGI. Details of this mechanism are given in the System Hardware section below.

Shareware Software

The concept of shareware is simple: release a product containing many useful features, but which has more advanced features and perhaps essential features limited, restricted, or locked out entirely, eg. being able to save files, or working on files over a particular size.

A user can download the shareware version of the software for free. They can test out the software and, if they like it, 'register' the software in order to obtain either the 'full' (ie. complete) version, or some kind of encrypted key or license code that will unlock the remaining features not accessible or present in the shareware version. Registration usually involves sending a small fee, eg. $30, to the author or company which created the software. Commonly, registration results in the author(s) sending the user proper printed and bound documentation, plus regular updates to the registered version, news releases on new features, access to dedicated mailing lists, etc.

The concept of shareware has changed over the years, partly due to the influence of the computer game 'Doom' which, although released as shareware in name, actually effectively gave away an entire third of the complete game for free. This was a ground-breaking move which proved to be an enormous success, earning the company which made the game (id Software, Dallas, Texas, USA) over eight million $US and a great deal of respect and loyalty from gaming fans. Never before had a company released shareware software in a form which did not involve deliberately 'restricting' key aspects of the shareware version. As stated above, shareware software is often altered so that,

for example, one could load files, work on them, make changes, test out a range of features, but (crucially) not save the results. Such shareware software is effectively not of any practical use on its own, ie. it serves only as a kind of hands-on advertisement for the full version. Doom was not like this at all. One could play an entire third of the game, including over a network against other players.

Today, other creative software designers have adopted a similar approach, perhaps the most famous recent example of which is 'Blender' [1], a free 3D rendering and animation program for UNIX and (as of very soon) WindowsNT systems.

In its as-supplied form, Blender can be used to do a great deal of work, creating 3D scenes, renderings and animations easily on a par with 3D Studio Max, even though some features in Blender are indeed locked out in the shareware version. However, unlike conventional traditional concepts, Blender does allow one to save files and so can be used for useful work. It has spread very rapidly in the last few months amongst students in educational sites worldwide, proving to be of particular interest to artists and animators who almost certainly could not normally afford a commercial package which might cost hundreds or perhaps thousands of pounds. Even small companies have begun using Blender.

However, supplied documentation for Blender is limited. As a 'professional level' system, it is unrealistic to expect to be able to get the best out of it without much more information on how it works and how to use it. Thus, the creators of Blender, a company called NaN based in Holland, makes most of their revenue by offering a very detailed 350 page printed and bound manual for about $50 US, plus a sequence of software keys which make available the advanced features in Blender.

Software distribution concepts such as the above methods used by NaN didn't exist just a few years ago, eg. before 1990. The rise of the Internet, certain games such as Doom, the birth of Linux, and changes in the way various UNIX vendors manage their business have caused a quantum leap in what people think of as shareware.

Note that the same caveat stated earlier with respect to software quality also applies to shareware, and to freeware too, ie. such software may or may not use the normal distribution method associated with a particular UNIX platform - in the case of SGIs, the 'inst' format.

Another famous example of shareware is the XV [2] image-viewer program, which offers a variety of functions for image editing and image processing (even though its author insists it's really just an image viewer). XV does not have restricted features, but it is an official shareware product which one is supposed to register if one intends to use the program for commercial purposes. However, as is typical with many modern shareware programs, the author stipulates that there is no charge for personal (non-commercial) or educational use.

Freeware Software

Unlike shareware software, freeware software is exactly that: completely free. There is no concept of registration, restricted features, etc. at all.

Until recently, even I was not aware of the vast amount of free software available for SGIs and UNIX systems in general. There always has been free software for UNIX systems, but as in keeping with other changes by UNIX vendors over the past few years, SGI altered its application

development support policy in 1997 to make it much easier for users to make use of freeware on SGI systems. Prior to that time, SGI did not make the system 'header' files (normally kept in /usr/include) publicly available. Without these header files, one could not compile any new programs even if one had a free compiler.

So, SGI adopted a new stance whereby the header files, libraries, example source code and other resources are provided free, but its own advanced compiler technologies (the MIPS Pro Compilers) remain commercial products. Immediately, anyone could then write their own applications for SGI systems using the supplied CDs (copies of which are available from SGI's ftp site) in conjunction with free compilation tools such as the GNU compilers. As a result, the 2nd-hand market for SGI systems in the USA has skyrocketed, with extremely good systems available at very low cost (systems which cost 37500 pounds new can now be bought for as little as 500 pounds, even though they can still be better than modern PCs in many respects).

It is highly likely that other vendors have adopted similar strategies in recent years (most of my knowledge concerns SGIs). Sun Microsystems made its SunOS free for students some years ago (perhaps Solaris too); my guess is that a similar compiler/development situation applies to systems using SunOS and Solaris as well - one can write applications using free software and tools. This concept probably also applies to HP systems, Digital UNIX systems, and other flavours of UNIX.

Linux is a perfect example of how the ideas of freeware development can determine an OS' future direction. Linux was meant to be a free OS from its very inception - Linus Torvalds, its creator, loathes the idea of an OS supplier charging for the very platform upon which essential software is executed. Although Linux is receiving considerable industry support these days, Linus is wary of the possibility of Linux becoming more commercial, especially as vendors such as Red Hat and Caldera offer versions of Linux with added features which must be paid for. Whether or not the Linux development community can counter these commercial pressures in order to retain some degree of freeware status and control remains to be seen.

Note: I'm not sure of the degree to which completely free development environments on a quality-par with GNU are available for MS Windows-based systems (whether that involves Win95, Win98, WinNT or even older versions such as Win3.1).

The GNU Licensing System

The GNU system is, without doubt, thoroughly unique in the modern era of copyright, trademarks, law suits and court battles. It can be easily summarised as a vast collection of free software tools, but the detail reveals a much deeper philosophy of software development, best explained by the following extract from the main GNU license file that accompanies any GNU-based program [3]:

> **"The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.**
>
> **When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.**

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all."

Reading the above extract, it is clear that those responsible for the GNU licensing system had to spend a considerable amount of time actually working out how to make something free! Free in a legal sense that is. So many standard legal matters are designed to *restrict* activities, the work put into the GNU Free Software Foundation makes the license document read like some kind of software engineer's nirvana. It's a serious issue though, and the existence of GNU is very important in terms of the unimaginable amount of creative work going on around the world which would not otherwise exist (without GNU, Linux would probably not exist).

SGI, and other UNIX vendors I expect, ships its latest OS (IRIX 6.5) with a CD entitled 'Freeware', which not only contains a vast number of freeware programs in general (everything from spreadsheets and data plotting to games, audio/midi programming and molecular modeling), but also a complete, pre-compiled inst-format distribution of the entire GNU archive: compilers, debugging tools, GNU versions of shells and associated utilities, calculators, enhanced versions of UNIX commands and tools, even higher-level tools such as a GUI-based file manager and shell tool, and an absolutely superb Photoshop-style image editing tool called GIMP [4] (GNU Image Manipulation Program) which is extendable by the user. The individual software subsystems from the Freeware CD can also be downloaded in precompiled form from SGI's web site [5].

The February 1999 edition of SGI's Freeware CD contains 173 different software subsystems, 29 of which are based on the GNU licensing system (many others are likely available from elsewhere on the Internet, along with further freeware items). A printed copy of the contents of the Feb99 Freeware CD is included with the course notes for further reading.

Other important freeware programs which are supplied separately from such freeware CD distributions (an author may wish to distribute just from a web site), include the Blue Moon Rendering Tools (BMRT) [6], a suite of advanced 3D ray-tracing and radiosity tools written by one of the chief architects at Pixar animation studios - the company which created "Toy Story", "Small Soldiers" and "A Bug's Life". Blender can output files in Inventor format, which can then be converted to RIB format for use by BRMT.

So why is shareware and freeware important? Well, these types of software matter because, today, it is perfectly possible for a business to operate using only shareware and/or freeware software. An increasingly common situation one comes across is an entrepreneurial multimedia firm using

Blender, XV, GIMP, BMRT and various GNU tools to manage its entire business, often running on 2nd-hand equipment using free versions of UNIX such as Linux, SunOS or IRIX 6.2! I know of one such company in the USA which uses decade-old 8-CPU SGI servers and old SGI workstations such as Crimson RealityEngine and IRIS Indigo. The hardware was acquired 2nd-hand in less than a year.

Whether or not a company decides to use shareware or freeware software depends on many factors, especially the degree to which a company feels it must have proper, official support. Some sectors such as government, medical and military have no choice: they must have proper, fully guaranteeable hardware and software support because of the nature of the work they do, so using shareware or freeware software is almost certainly out of the question. However, for medium-sized or smaller companies, and especially home users or students, the existence of shareware and freeware software, combined with the modern approaches to these forms of software by today's UNIX vendors, offers whole new avenues of application development and business ideas which have never existed before as commercially viable possibilities.

System Hardware

The hardware platforms supplied by the various UNIX vendors are, like UNIX itself today, also designed and managed with a top-down approach.

The world of PCs has always been a bottom-up process of putting together a mish-mash of different components from a wide variety of sources. Motherboards, video cards, graphics cards and other components are available in a plethora of types of varying degrees of quality. This bottom-up approach to systems design means it's perfectly possible to have a PC with a good CPU, good graphics card, good video card, but an awful motherboard. If the hardware is suspect, problems faced by the user may appear to be OS-related when in fact they could be down to poor quality hardware. It's often difficult or impossible to ascertain the real cause of a problem - sometimes system components just don't work even though they should, or a system suddenly stops recognising the presence of a device; these problems are most common with peripherals such as CDROM, DVD, ZIP, sound cards, etc.

Dealing only with hardware systems designed specifically to run a particular vendor's UNIX variant, the situation is very different. The vendor maintains a high degree of control over the design of the hardware platform. Hence, there is opportunity to focus on the unique requirements of target markets, quality, reliability, etc. rather than always focusing on absolute minimum cost which inevitably means cutting corners and making tradeoffs.

This is one reason why even very old UNIX systems, eg. multi-processor systems from 1991 with (say) eight 33MHz CPUs, are still often found in perfect working order. The initial focus on quality results in a much lower risk of component failure. Combined with generous hardware and software support policies, hardware platforms for traditional UNIX systems are far more reliable than PCs.

My personal experience is with hardware systems designed by SGI, about which I know a great deal. Their philosophy of design is typical of most UNIX hardware vendors (others would be Sun, HP, IBM, DEC, etc.) and can be contrasted very easily with the way PCs are designed and constructed:

```
UNIX low-end:   "What can we give the customer for 5000?"
     mid-range: "What can we give the customer for 15000?"
     high-end:  "What can we give the customer for 65000+?"
```

```
PC: "How cheap can we make a machine which offers
     a particular feature set and level of ability?"
```

Since the real driving force behind PC development is the home market, especially games, the philosophy has always been to decide what features a typical 'home' or 'office' PC ought to have and then try and design the cheapest possible system to offer those features. This approach has eventually led to incredibly cut-throat competition, creating new concepts such as the 'sub-$1000' PC, and even today's distinctly dubious 'free PC', but in reality the price paid by consumers is the use of poor quality components which do not integrate well, especially components from different suppliers. Hardware problems in PCs are common, and now unavoidable. In Edinburgh, I know of a high-street PC store which *always* has a long queue of customers waiting to have their particular problem dealt with.

By contrast, most traditional UNIX vendors design their own systems with a top-down approach which focuses on quality. Since the vendor usually has complete control, they can ensure a much greater coherence of design and degree of integration. System components work well with each other because all parts of the system were designed with all the other parts in mind.

Another important factor is that a top-down approach allows vendors to innovate and develop new architectural designs, creating fundamentally new hardware techniques such as SMP and S2MP processing, highly scalable systems, advanced graphics architectures, and perhaps most importantly of all from a customer's point of view: much more advanced CPU designs (Alpha, MIPS, SPARC, PA-RISC, POWER series, etc.) Such innovations and changes in design concept are impossible in the mainstream PC market: there is too much to lose by shifting from the status-quo. Everything follows the lowest common denominator.

The most obvious indication of these two different approaches is that UNIX hardware platforms have always been more expensive than PCs, but that is something which should be expected given that most UNIX platforms are deliberately designed to offer a much greater feature set, better quality components, better integration, etc.

A good example is the SGI Indy. With respect to absolute cost, the Indy was very expensive when it was first released in 1993, but because of what it offered in terms of hardware and software features it was actually a very cheap system compared to trying to put together a PC with a similar feature set. In fact, Indy offered features such as hardware-accelerated 3D graphics at high resolution (1280x1024) and 24bit colour at a time when such features did not exist at all for PCs.

PCW magazine said in its original review [7] that to give a PC the same standard features and abilities, such as ISDN, 4-channel 16bit stereo sound with multiple stereo I/O sockets, S-Video/Composite/Digital video inputs, NTSC-resolution CCD digital camera, integrated SCSI, etc. would have cost twice as much as an Indy. SGI set out to design a system which would include all these features as-standard, so the end result was bound to cost several thousand pounds, but that was still half the cost of trying to cobble together a collection of mis-matched components from a dozen different companies to produce something which still would not have been anywhere near as good. As PCW put it, the Indy - for its time - was a great machine offering superb value *if* one was the kind of customer which needed its features and would be able to make good use of them.

Sun Microsystems adopted a similar approach to its recent Ultra5, Ultra10 and other systems: provide the user with an integrated design with a specific feature set that Sun knew its customers wanted. SGI did it again with their O2 system, released in October 1996. O2 has such a vast range of features (highly advanced for its time) that few ordinary customers would find themselves using most or all of them. However, for the intended target markets (ranging from CAD, design,

animation, film/video special effects, video editing to medical imaging, etc.) the O2 was an excellent system. Like most UNIX hardware systems, O2 today is not competitive in certain areas such as basic 3D graphics performance (there are exceptions to this), but certain advanced and unique architectural features mean it's still purchased by customers who require those features.

This, then, is the key: UNIX hardware platforms which offer a great many features and high-quality components are only a good choice if one:

- is the kind of customer which definitely needs those features
- values the ramifications of using a better quality system that has been designed top-down: reliability, quality, long-term value, ease of maintenance, etc.

One often observes people used to PCs asking why systems like O2, HP's Visualize series, SGI's Octane, Sun's Ultra60, etc. cost so much compared compared to PCs. The reason for the confusion is that the world of PCs focuses heavily on the abilities of the main CPU, whereas all UNIX vendors have, for many years, made systems which include as much dedicated acceleration hardware as possible, easing the burden on the main CPU. For the home market, systems like the Amiga pioneered this approach; unfortunately, the company responsible fort the Amiga doomed itself to failure as a result of various marketing blunders.

From an admin's point of view, the practical side effect of having to administer and run a UNIX hardware platform is that there is far, far less effort needed in terms of configuring systems at the hardware level, or having to worry about different system hardware components operating correctly with one other. Combined with the way most UNIX variants deal with hardware devices (ie. automatically and transparently most of the time), a UNIX admin can swap hardware components between different systems from the same vendor without any need to alter system software, ie. any changes in system hardware configuration are dealt with automatically.

Further, many UNIX vendors use certain system components that are identical (usually memory, disks and backup devices), so admins can often swap generic items such as disks between different vendor platforms without having to reconfigure those components (in the case of disks) or worry about damaging either system. SCSI disks are a good example: they are supplied preformatted, so an admin should never have to reformat a SCSI disk. Swapping a SCSI disk between different vendor platforms may require repartitioning of the disk, but never a reformat. In the 6 years I've been using SGIs, I've never had to format a SCSI disk.

Examining a typical UNIX hardware system such as Indy, one notices several very obvious differences compared to PCs:

- There are far fewer cables in view.
- Components are positioned in such a way as to greatly ease access to all parts of the system.
- The overall design is highly integrated so that system maintenance and repairs/replacements are much easier to carry out.

Thus, problems that are solvable by the admin can be dealt with quickly, while problems requiring vendor hardware support assistance can be fixed in a short space of time by a visiting technician, which obviously reduces costs for the vendor responsible by enabling their engineers to deal with a larger number of queries in the same amount of time.

Just as with the approaches taken to hardware and software design, the way in which support contracts for UNIX systems operate also follow a top-down approach. Support costs can be high, but the ethos is similar: you get what you pay for - fast no-nonsense support when it's needed.

I can only speak from experience of dealing with SGIs, but I'm sure the same is true of other UNIX vendors. Essentially, if I encounter a hardware problem of some kind, the support service always errs on the side of caution in dealing with the problem, ie. I don't have to jump through hoops in order to convince them that there is a problem - they accept what I say and organise a visiting technician to help straight away (one can usually choose between a range of response times from 1 hour to 5 days). Typically, unless the technician can fix the problem on-site in a matter of minutes, then some, most, or even all of the system components will be replaced if necessary to get the system in working order once more.

For example, when I was once encountering SCSI bus errors, the visiting engineer was almost at the point of replacing the motherboard, video card and even the main CPU (several thousand pounds worth of hardware in terms of new-component replacement value at the time) before some extra further tests revealed that it was in fact my own personal disk which was causing the problem (I had an important jumper clip missing from the jumper block). In other words, UNIX vendor hardware support contracts tend to place much less emphasis on the customer having to prove they have a genuine problem.

I should imagine this approach exists because many UNIX vendors have to deal with extremely important clients such as government, military, medical, industrial and other sectors (eg. safety critical systems). These are customers with big budgets who don't want to waste time messing around with details while their faulty system is losing them money - they expect the vendor to help them get their system working again as soon as possible.

Note: assuming a component is replaced (eg. motherboard), even if the vendor's later tests show the component to be working correctly, it is not returned to the customer, ie. the customer keeps the new component. Instead, most vendors have their own dedicated testing laboratories which pull apart every faulty component returned to them, looking for causes of problems so that the vendor can take corrective action if necessary at the production stage, and learn any lessons to aid in future designs.

To summarise the above:

- A top-down approach to hardware design means a better feature set, better quality, reliability, ease of use and maintenance, etc.

- As a result, UNIX hardware systems can be costly. One should only purchase such a system if one can make good use of the supplied features, and if one values the implications of better quality, etc., despite the extra cost.

However, a blurred middle-ground between the top-down approach to UNIX hardware platforms and the bottom-up approach to the supply of PCs is the so-called 'vendor-badged' NT workstation market. In general, this is where UNIX vendors create PC-style hardware systems that are still based on off-the-shelf components, but occasionally include certain modifications to improve performance, etc. beyond what one normally sees of a typical PC. The most common example is where vendors such as Compaq supply systems which have two 64bit PCI busses to increase available system bandwidth.

All these systems are targeted at the 'NT workstation' market. Cynics say that such systems are just a clever means of placing a 'quality' brand name on ordinary PC hardware. However, such systems do tend to offer a better level of quality and integration that ordinary PCs (even expensive ordinary PCs), but an inevitable ironic side effect is that these vendor-badged systems do cost more. Just as with traditional UNIX hardware systems, whether or not that cost is worth it depends on customers' priorities. Companies such as movie studios regard stability and reliability as absolutely critical, which is why most studios do not use NT [8]. Those that do, especially smaller studios (perhaps because of limited budgets) will always go for vendor-badged NT workstations rather than purchasing systems from PC magazines and attempting to cobble together a reliable platform. The extra cost is worth it.

There is an important caveat to the UNIX hardware design approach: purchasing what can be a very good UNIX hardware system is a step that can easily be ruined by not equipping that system in the first instance with sufficient essential system resources such as memory capacity, disk space, CPU power and (if relevant) graphics/image/video processing power. Sometimes, situations like this occur because of budget constraints, but the end result may be a system which cannot handle the tasks for which it was purchased. If such mis-matched purchases are made, it's usually a good sign that the company concerned is using a bottom-up approach to making decisions about whether or not to buy a hardware platform that has been built using a top-down approach. The irony is plain to see. Since admins often have to advise on hardware purchases or upgrades, a familiarity with these issues is essential.

Conclusion: decide what is needed to solve the problem. Evaluate which systems offer appropriate solutions. If there no system is affordable, do not compromise on essentials such as memory or disk as a means of lowering cost - choose a different platform instead such as good quality NT system, or a system with lower costs such as an Intel machine running Linux, etc.

Similarly, it makes no sense to have a good quality UNIX system, only to then adopt a strategy of buying future peripherals (eg. extra disks, memory, printers, etc.) that are of poor quality. In fact, some UNIX vendors may not offer or permit hardware support contracts unless the customer sticks to using approved 3rd-party hardware sources.

Summary: UNIX hardware platforms are designed top-down, offer better quality components, etc., but tend to be more expensive as a result.

Today, an era when even SGI has started to sell systems that support WindowsNT, the philosophy is still the same: design top-down to give quality hardware, etc. Thus, SGI's WindowsNT systems *start* at around 2500 pounds - alot by the standards of any home user, but cheap when considering the market in general. The same caveat applies though: such a system with a slow CPU is wasting the capabilities of the machine.

UNIX Characteristics.

Integration:

A top-down approach results in an integrated design. Systems tend to be supplied 'complete', ie. everything one requires is usually supplied as-standard. Components work well together since the designers are familiar with all aspects of the system.

Stability and Reliability:

The use of quality components, driven by the demands of the markets which most UNIX vendors aim for, results in systems that experience far fewer component failures compared to PCs. As a result of a top-down and integrated approach, the chances of a system experiencing hardware-level conflicts are much lower compared to PCs.

Security:

It is easy for system designers to incorporate hardware security features such as metal hoops that are part of the main moulded chassis, for attaching to security cables.

On the software side, and as an aid to preventing crime (as well as making it easier to solve crime in terms of tracing components, etc.) systems such as SGIs often incorporate unique hardware features. The following applies to SGIs but is also probably true of hardware from other UNIX vendors in some equivalent form.

Every SGI has a PROM chip on the motherboard, without which the system will not boot. This PROM chip is responsible for initiating the system bootup sequence at the very lowest hardware level. However, the chip also contains an ID number which is unique to that particular machine. One can display this ID number with the following command:

```
sysinfo -s
```

Alternatively, the number can be displayed in hexadecimal format by using sysinfo command on its own (one notes the first 4 groups of two hex digits). A typical output might look like this:

```
% sysinfo -s
1762299020
% sysinfo
System ID:
69  0a  8c  8c  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
```

The important part of the output from the second command is the beginning sequence consisting of ’690A8C8C’.

The ID number is not only used by SGI when dealing with system hardware and software support contracts, it is also the means by which license codes are supplied for SGI’s commercial software packages.

If one wishes to use a particular commercial package, eg. the VRML editor called CosmoWorlds, SGI uses the ID number of the machine to create a license code which will be recognised by the program concerned as being valid *only* for that particular machine. The 20-digit hexadecimal license code is created using a special form of encryption, presumably combining the ID number with some kind of internal database of codes for SGI’s various applications which only SGI has access to. In the case of the O2 I use at home, the license code for CosmoWorlds is 4CD4FB82A67B0CEB26B7 (ie. different software packages on the same system need different license codes). This code will not work for any other software package on any other SGI anywhere in the world.

There are two different license management systems in use by SGIs: the NetLS environment on older platforms, and the FlexLM environment on newer platforms. FlexLM is being widely adopted by many UNIX vendors. NetLS licenses are stored in the /var/netls directory, while FlexLM licenses are kept in /var/flexlm. To the best of my knowledge, SGI's latest version of IRIX (6.5) doesn't use NetLS licenses anymore, though it's possible that 3rd-party software suppliers still do.

As stated in the software section, the use of the ID number system at the hardware level means it is impossible to pirate commercial software. More accurately, anyone can copy any SGI software CD, and indeed install the software, but that software will not run without the license code which is unique to each system, so there's no point in copying commercial software CDs or installing copied commercial software in the first place.

Of course, one could always try to reverse-engineer the object code of a commercial package to try and get round the section which makes the application require the correct license code, but this would be very difficult. The important point is that, to the best of my knowledge, SGI's license code schema has never been broken at the hardware level.

Note: from the point of view of an admin maintaining an SGI system, if a machine completely fails, eg. damage by fire and water, the admin should always retain the PROM chip if possible - ie. a completely new system could be obtained but only the installation of the original PROM chip will make the new system effectively the same as the old one. For PCs, the most important system component in terms of system identity is the system disk (more accurately, its contents); but for machines such as SGIs, the PROM chip is just as if not more important than the contents of the system disk when it comes to a system having a unique identity.

Scalability.

Because a top-down hardware design approach has been used by all UNIX hardware vendors over the years, most UNIX vendors offer hardware solutions that scale to a large number of processors. Sun, IBM, SGI, HP and other vendors all offer systems that scale to 64 CPUs. Currently, one cannot obtain a reliable PC/NT platform that scales to even 8 CPUs (Intel won't begin shipping 8-way chip sets until Q3 1999).

Along with the basic support for a larger number of processors, UNIX vendors have spent a great deal of time researching advanced ways of properly supporting many CPUs. There are complex issues concerning how such systems handle shared memory, the movement of data, communications links, efficient use of other hardware such as graphics and video subsystems, maximised use of storage systems (eg. RAID), and so on.

The result is that most UNIX vendors offer large system solutions which can tackle extremely complex problems. Since these systems are obviously designed to the very highest quality standards with a top-down approach to integration, etc., they are widely used by companies and institutions which need such systems for solving the toughest of tasks, from processing massive databases to dealing with huge seismic data sets, large satellite images, complex medical data and intensive numerical processing (eg. weather modeling).

One very beneficial side-effect of this kind of development is that the technology which comes out of such high-quality designs slowly filters down to the desktop systems, enabling customers to eventually utilise extremely advanced and powerful computing systems. A particularly good example of this is SGI's Octane system [9] - it uses the same components and basic technology as

SGI's high-end Origin server system. As a result, the user benefits from many advanced features, eg.

- Octane has no inherent maximum memory limit. Memory is situated on a 'node board' along with the 1 or 2 main CPUs, rather than housed on a backplane. As CPU designs improve, so memory capacity on the node board can be increased by using a different node board design, ie. without changing the base system at all. For example, Octane systems using the R10000 CPU can have up to 2GB RAM, while Octane systems using the R12000 CPU can have up to 4GB RAM. Future CPUs (R14K, R16K, etc.) will change this limit again to 8GB, 16GB, etc.

- The speed at which all internal links operate is directly synchronised to the clock speed of the main CPU. As a result, internal data pathways can always supply data to both main CPUs faster than they can theoretically cope with, ie. one can get the absolute maximum performance out of a CPU (this is fundamentally not possible with any PC design). As CPU clock speeds increase, so does the rate at which the system can move data around internally. An Octane using 195MHz R10000s offers three separate internal data pathways each operating at 1560MB/sec (10X faster than a typical PCI bus). An Octane using 300MHz R12000s runs the same pathways at the faster rate of 2400MB/sec per link. ie. system bandwidth and memory bandwidth increase to match CPU speed.

The above is not a complete list of advanced features.

SGI's high-end servers are currently the most scalable in the world, offering up to 256 CPUs for a commercially available system, though some sites with advance copies of future OS changes have systems with 512 and 720 CPUs. As stated elsewhere, one system has 6144 CPUs.

The quality of design required to create technologies like this, along with software and OS concepts that run them properly, are quite incredible. These features are passed on down to desktop systems and eventually into consumer markets. But it means that, at any one time, mid-range systems based on such advanced technologies can be quite expensive (Octanes generally start at around 7000 pounds). Since much of the push behind these developments comes from military and government clients, again there is great emphasis on quality, reliability, security, etc. Cray Research, which is owned by SGI, holds the world record for the most stable and reliable system: a supercomputer with 2048 CPUs which ran for 2.5 years without any of the processors exhibiting a single system-critical error.

Sun, HP, IBM, DEC, etc. all operate similar design approaches, though SGI/Cray happens to have the most advanced and scalable server and graphics system designs at the present time, mainly because they have traditionally targeted high-end markets, especially US government contracts.

The history of UNIX vendor CPU design follows a similar legacy: typical customers have always been willing to pay 3X as much as an Intel CPU in order to gain access to 2X the performance. Ironically, as a result, Intel have always produced the world's slowest CPUs, even though they are the cheapest. CPUs at much lower clock speeds from other vendors (HP, IBM, Sun, SGI, etc.) can easily be 2X to 5X faster than Intel's current best. As stated above though, these CPUs are much more expensive - even so, it's an extra cost which the relevant clients say they will always bare in order to obtain the fastest available performance. The exception today is the NT workstation market where systems from UNIX vendors utilise Intel CPUs and WindowsNT (and/or Linux), offering a means of gaining access to better quality graphics and video hardware while sacrificing the use of more powerful CPUs and the more sophisticated UNIX OSs, resulting in lower cost. Even so, typical high-end NT systems still cost around 3000 to 15000 pounds.

So far, no UNIX vendor makes any product that is targeted at the home market, though some vendors create technologies that are used in the mass consumer market (eg. the R3000 CPU which runs the Sony PlayStation is designed by SGI and was used in their older workstations in the late 1980s and early 1990s; all of the Nintendo64's custom processors were designed by SGI). In terms of computer systems, it is unlikely this situation will ever change because to do so would mean a vendor would have to adopt a bottom-up design approach in order to minimise cost above all else - such a change wouldn't be acceptable to customers and would contradict the way in which the high-end systems are developed. Vendors which do have a presence in the consumer market normally use subsidiaries as a means of avoiding internal conflicts in design ethos, eg. SGI's MIPS subsidiary (soon to be sold off).

References:

1. Blender Animation and Rendering Program:

   `http://www.blender.nl/`

2. XV Image Viewer:

   `http://www.trilon.com/xv/xv.html`

3. Extract taken from GNU GENERAL PUBLIC LICENSE, Version 2, June 1991, Copyright (C) 1989, 1991 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

4. GIMP (GNU Image Manipulation Program):

   `http://www.gimp.org/`

5. SGI Freeware Sites (identical):

   `http://freeware.sgi.com/`
   `http://toolbox.sgi.com/TasteOfDT/public/freeware/`

6. Pixar's Blue Moon Rendering Tools (BMRT):

   `http://www.bmrt.org/`

7. Silicon Graphics Indy, PCW, September 1993:

   `http://www.futuretech.vuurwerk.nl/pcw9-93indy.html`

8. "LA conferential", CGI Magazine, Vol4, Issue 1, Jan/Feb 1999, pp. 21, by Richard Spohrer.

   Interview from the 'Digital Content and Creation' conference and exhibition:

   > '"No major production facilities rely on commercial software, everyone has to customise applications in order to get the most out of them," said Hughes. "We run Unix on SGI as we need a stable environment which allows fast networking. NT is not a professional solution and was never designed to handle high-end network environments," he added. "Windows NT is the antithesis of what the entertainment industry needs. If we were to move from Irix, we would use Linux over NT."'

- John Hughes, president/CEO of Rhythm & Hues and Scott Squires, visual effects supervisor at ILM and ceo of Puffin Design.

9. Octane Information Index:

   `http://www.futuretech.vuurwerk.nl/octane/`

10. "How to set up the BIND domain name server", Network Week, Vol4 No. 29, 14th April 1999, pp. 17, by David Cartwright.

11. A letter from a reader in response to [10]:

    "Out of a BIND", Network Week, Vol4 No. 31, 28th April 1999, pp. 6:

    "A couple of weeks ago, I had a problem. I was attempting to configure NT4's DNS Server for use on a completely private network, but it just wasn't working properly. The WindowsNT 'help' - and I use that term loosely - assumed my network was connected to the Internet, so the examples it gave were largely useless. Then I noticed David Cartwright's article about setting up DNS servers. (Network Week, 14th April). The light began to dawn. Even better, the article used BIND's configuration files as examples. This meant that I could dump NT's obtuse GUI DNS Manager application and hand-hack the configuration files myself. A few minor problems later (most of which were caused by Microsoft's example DNS config files being a bit... um... optimistic) and the DNS server finally lurched into life. Thank you Network Week. The more Q&A and how-to type information you print, the better."

    - Matthew Bell, Fluke UK.

General References:

```
Anonymous SGI FTP Site List:  http://reality.sgi.com/billh/anonftp/
Origin2000 Information Index: http://www.futuretech.vuurwerk.nl/origin/
Onyx2 Information Index:      http://www.futuretech.vuurwerk.nl/onyx2/
SGI:                          http://www.sgi.com/
Hewlett Packard:              http://www.hp.com/
Sun Microsystems:             http://www.sun.com/
IBM:                          http://www.ibm.com/
Compaq/Digital:               http://www.digital.com/
SCO:                          http://www.sco.com/
Linux:                        http://www.linux.org/
```

Appendix A: Case Study.

For unknown and unchangeable reasons, UCLAN's central admin system has a DNS setup which, incorrectly, does not recognise comp.uclan.ac.uk as a subdomain. Instead, the central DNS lists comp as a host name, ie. comp.uclan.ac.uk is listed as a direct reference to Yoda's external IP address, 193.61.250.34; in terms of the intended use of the word 'comp', this is rather like referring to a house on a street by using just the street name. As a result, the SGI network's fully qualified host names, such as yoda.comp.uclan.ac.uk, are not recognised outside UCLAN, and neither is comp.uclan.ac.uk since all the machines on the SGI network treat comp as a subdomain. Thus, external users can access Yoda's IP address directly by referring to 193.61.250.34 (so ftp is

possible), but they cannot access Yoda as a web server, or access individual systems in Ve24 such as sevrin.comp.uclan.ac.uk, or send email to the SGI network. Also, services such as USENET cannot be setup, so internal users must use web sites to access newsgroups.

This example serves as a warning: organisations should thoroughly clarify what their individual department's network structures are going to be, through a proper consultation and discussion process, *before* allowing departments to setup internal networks. Otherwise, confusion and disagreement can occur. In the case of the SGI network, its internal structure is completely correct (as confirmed by SGI themselves), but the way it is connected to the Internet is incorrect. Only the use of a Proxy server allows clients to access the Internet, but some strange side-effects remain; for example, email can be sent from the SGI network to anywhere on the Internet (from Yoda to Yahoo in less than 10 seconds!), but not vice-versa because incoming data is blocked by the incorrectly configured central DNS.

Email from the SGI network can reach the outside world because of the way the email system works: the default settings installed along with the standard Berkeley Sendmail software (/usr/lib/sendmail) are sufficient to forward email from the SGI network to the Internet via routers further along the communications chain, which then send the data to JANET at Manchester, and from there to the final destination (which could include a UCLAN student or staff member). The situation is rather like posting a letter without a sender's address, or including an address which gives everything as far as the street name but not the house number - the letter will be correctly delivered, but the recipient will not be able to reply to the sender.