Why the Plan 9® Distributed System Matters (DRAFT of 29 December 2003)

Geoff Collyer

ABSTRACT

This paper was the start of an outline for a proposed book. I've been told that there's no audience for it by a major publisher; judge for yourself. I expect that the audience will be people who are familiar with computers, probably users of UNIX® or Linux.

1. Orientation

I am about to argue for Plan 9 and at least implicitly against other systems. I could preface negative evaluations of aspects of other systems with ''Let's be honest; you know it's really true that ...'' to try to soften the blows, but I'd rather you read what follows with that in mind. If we're honest, our existing systems have lots of problems. We won't find better solutions by ignoring those problems. I have tried to write without being too harsh; where I seem to be harsh, I'm trying to emphasise the biggest problems. Somebody has to say ''the emperor **really** has no clothes'' while most other people are applauding the lovely invisible clothes, and in this case, that person is me. Onward ...

This is also not about 'my operating system can beat up your operating system'. I've lived on Plan 9 almost exclusively since 1995 (and sporadically before that) and found it to be comfortable and productive, and I think many other people would too if they took a serious look at Plan 9. I had previously been a happy user of Inferno, UNIX (many flavours), TOPS-10, and systems so primitive that they had no real operating systems, just program loaders. I've also been an unhappy casual user of some other systems. I have even been known to enter (small) programs through the front-panel lights and switches, so I like to think that my judgement is not wholly uninformed. It's easy to grow attached to the first system (or language) you use. By the time I encountered UNIX, I'd used a few systems, but UNIX blew me away. The next time I had that experience was when I encountered Plan 9, and by then I'd seen a lot of systems. When I criticise UNIX (and Linux), it's from first-hand experience running UNIX systems and programming them, including modifying UNIX kernels, over many years. (I have a convenient shorthand, ''lunix'' (pronounced loo-niks), which I use to mean ''UNIX, Linux or both''.)

2. Introduction

Plan 9 from Bell Labs[26] is a distributed operating system, created at Bell Labs in the same research center, and by some of the same individuals, who created and evolved UNIX[33], but with twenty years experience with UNIX to draw on. It has been available to anyone for US$350 since its second edition in 1995 and for free via the Web since its third edition in 2001, yet has had relatively little adoption, certainly not the widespread impact that UNIX has had. The distribution includes complete source code, documentation and Intel x86 binaries.

Plan 9 is not just another free UNIX-like system, such as the free BSDs or Linux, and deserves more exposure (and wider adoption) than it has seen so far. Plan 9 is not a UNIX port, nor a UNIX clone; it's more like UNIX re-thought, redesigned, and re-implemented with over twenty years of experience. Plan 9 is a new system, re-implemented from the ground up. Some UNIX commands have been adapted (for example, to use the Unicode character set and UTF-8 byte encoding of Unicode)[8, 37, 38], but Plan 9 really is a new system. Anyone familiar with the evolution of Research UNIX[23, 20, 7] beyond Seventh Edition[21] shouldn't be startled by much in Plan 9, but Plan 9 has evolved considerably.

A later distributed system from Bell Labs, Inferno[10, 3], is derived from the ideas in Plan 9, and its kernel shares much code with the Plan 9 kernel, but all code outside the kernel is written in Limbo[34, 11], a safer (but more restricted) language than C[32] that borrows from C, Pascal[17, 18] and CSP[16]. Limbo might be

Copyright © 2003 Geoff Collyer

compared with Java:[15] it compiles to byte code, which is then interpreted or compiled on-the-fly, it performs automatic garbage collection, its character set is Unicode, etc., but I find Limbo's support for concurrent programming to be more convenient than Java's. Limbo's threads are not pre-empted and communication among threads is primarily via messages on channels. There is plenty of locking done, but it's within the Limbo run-time support (e.g., serialising access to channels), not directly visible to the programmer.

3. Why Should We Care About Plan 9?

Plan 9 is a general-purpose, multi-user, portable distributed system implemented on a variety of computers and networks. It lacks a number of features often found in other distributed systems, including

(i) *A uniform distributed name space,*
(ii) *Process migration,*
(iii) *Lightweight processes,*
(iv) *Distributed file caching*
(v) *Personalised workstations,*
(vi) *Support for X windows.*

– *Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey in ''Plan 9 from Bell Labs'', 1993.*

Plan 9 demonstrates that we can do better than (non-Research) UNIX[1,2], which never really adapted very well to modern networks and graphics. In truth, UNIX has also accumulated an awful lot of rather crufty ideas and code over the years. Linux is just a clone of UNIX, except that there is rather less quality control of the code. Plan 9 is also a delight to use.

Plan 9 is available for free and its licence explicitly permits commercial use. The distribution includes full sources and documentation. One can recompile all the binaries in the system, including compiling for different architectures. As distributed, Plan 9 runs on Intel x86 PCs and clones, DEC/Compaq/HP ''AlphaPCs'', Compaq/HP Ipaq 3600s, and some Power PC evaluation boards. Only peripherals for which Plan 9 drivers exist are supported out of the box. Plan 9 has run on SGIs, Suns and NeXTs in the past, and a couple of Sun ports are available from the Plan 9 web site, `http://plan9.bell-labs.com/plan9dist`.

4. The Users' and Administrators' Perspective

4.1. Simpler Administration

Most Plan 9 systems other than file servers are run diskless, or their disks are used only for bootstrapping. Given modern 100base-T (or faster) switched Ethernets, cheap RAM, and Plan 9's remote file system protocol, which permits caching, this works much better than you may remember diskless Suns working in the 1980's.

Adding a new machine is a matter of adding a few lines to `/lib/ndb/local`, plugging the machine into the appropriate Ethernet, and turning it on. PCs also need to run a Plan 9 bootstrap program since their boot ROMs, uniquely among modern computers, often do not implement booting over the network via BOOTP (or DHCP) and TFTP.

☞ **WARNING:** It's time to be brutally honest again. Take a deep breath. ☜ UNIX's name server, BIND[6], really is a security hazard (look at the CERT alerts for it) and awkward and error-prone to configure (it has its own O'Reilly book). Instead, Plan 9 has *dns* (in *ndb*(8)), which permits specification of forward and reverse maps in one place, with no duplication of data. *Dns* also generates zone serial numbers automatically. These two conveniences cure the two most common DNS problems: inconsistent maps, and forgetting to bump the serial number up, which causes the updated zone data to not propagate.

☞ **WARNING:** It's time to be brutally honest again. Take a deep breath. ☜ UNIX's *sendmail*[4] **really is** big (~80,000 lines of C at last count), slow, buggy, difficult to configure correctly (it has its own fat O'Reilly book) and famously insecure (look at the *many* CERT alerts for it, many of them buffer overruns). Instead, Plan 9's mail system, *upas* (see *mail*(1))[29], is smaller (~23,500 lines) despite doing more (such as *upasfs*(4)), fast, straight-forward to configure, secure (for example, it uses dynamically-sized strings everywhere, so buffer overruns are not possible), fast and well-written.

Freedom from BIND, *sendmail*, and local state on each machine are major wins for system administrators. Terminals, also known as workstations, can be utterly interchangeable, allowing one to work anywhere there is a terminal, which includes laptops and Ipaqs.

4.2. Specialisation

One of the key ideas behind Plan 9 is that specialised systems can be more effective than completely general-purpose ones. Running the same large UNIX kernel and daemons on all one's Suns, for example, does not make good use of their resources (and probably isn't good security practice either). A typical non-trivial Plan 9 installation consists of one or more file servers, one or more CPU servers (compute servers) and some collection of terminals.

Collecting the permanent storage in one place makes it easier to backup and to find. The file servers implement daily snapshots of all permanent files back to the beginning of a given file server's operation, while recording only the blocks that change each night. Having these snapshots conveniently available permits restoring ones own files without administrative assistance, and that in turn encourages experimentation, knowing that yesterday's version of a file can be quickly retrieved if needed. The snapshots also provide a simple form for revision control. File servers are normally limited to just file service, so that they continue to provide fast service.

Shared CPU servers normally run all the time, so they provide convenient places to offer services, such as listening for incoming mail, DNS, *cpu*(1), and *ssh* requests. They also tend to be configured with a lot of RAM and multiple fast processors, so they are good places to perform heavy computation, including compilation. They may also have faster network connections to the file server(s) than the terminals have. Upgrading the CPU servers improves the computing base of an entire Plan 9 installation.

Terminals may be small machines by the standards of Microsoft Windows. They typically run one or more instances of the window system, *rio*(4)[27], plus editors, VNC clients, etc. It is possible to perform on a terminal any computation that could be done on a CPU server, but it usually makes more sense to do heavy computing on a CPU server. Conversely, one can run editors, window systems, etc. on the CPU servers, with local display and with reasonable interactive response, but one gets even better response from running them locally on one's terminal.

4.3. Universal Character Set and Encoding

Plan 9 uses a single character set, Unicode, throughout. Unicode covers virtually all the world's scripts, including some dead ones. To remain compatible with existing ASCII files, Unicode data are encoded externally (in files, pipes and network connections) using the UTF-8 encoding. There is thus no need to switch encodings, as in Microsoft Windows; one character set and one encoding suffice.

Even file names may contain UTF-encoded characters: ☺◊逃 is a legal file name.

Characters from any script can co-exist within a single file, like this:

はGeoff Collyerが開発したデー
タ・ベース(news overview database)であるNOVをサポートしています

4.4. Better Graphics and Window System(s)

☞ **WARNING:** It's time to be brutally honest again. Take a deep breath. ☜ *X11* is one of the three worst UNIX components (the other two being *sendmail* and BIND). All three really should have been replaced by now. X11 is a poor window system: big (X usually occupies half of the disk space of a UNIX distribution and half of the RAM of a running UNIX system), unresponsive if there's the least bit of network or CPU load, complex, and hard to write clients for. The many X toolkits (libraries) try to make it easier to write clients by layering on more code, but they mostly don't help much.

Plan 9 provides graphics identically with or without multiplexing (a window system). The window system, *rio*, is small, intuitive to use and relatively easy to program, as are the graphics[24, 25]. (I wrote a

Copyright © 2003 Geoff Collyer

Plan 9 graphics front-end for a Reversi game; it's 469 lines of C. I have never made the attempt with X, it's so daunting.) *Rio* is also very responsive, even when run remotely. Clicking a mouse button results in immediate action, not hesitation, perhaps followed at some later date by action. One can use *rio* windows to connect to non-Plan-9 systems, notably via *ssh*, though there are other, more interesting possibilities described later.

Although the *ed* and *sed* editors are available, the most commonly-used editor is *sam*(1), a multi-file, multi-window mouse-based editor which runs as two processes, editor and display, connected via a pipe or network connection. *Sam* ports for UNIX and Microsoft Windows exist, and it is possible to run the editor process on a remote non-Plan-9 system with the display running locally. This is a particularly good way to edit remotely when bandwidth is scarce.

There is also an alternative user interface, *acme*(4)[28], that some people use as their editor.

4.5. Better Networking

☞ **WARNING:** It's time to be brutally honest again. Take a deep breath. ☜ Sockets have convinced most people that network programming is complicated and hard; it needn't be. Mainstream UNIX networking was copied at the University of California at Berkeley from TOPS-20 and Tenex (early DEC PDP-10 ARPAnet operating systems) into UNIX without much thought for the appropriateness of the Twenex mechanisms to UNIX, or even whether they were good mechanisms to start with. So normal UNIX network applications are festooned with arcane details of particular network protocols and their bizarre implementations.

Plan 9 is a distributed system, so networking is central to it. Its networking code has to be fast, reliable and easy to use. Somewhat more thought has gone into networking in Plan 9. All resources in Plan 9 are accessed via the file system namespace. Network interfaces (e.g., Ethernet cards) are in-kernel file servers that each serve a small directory tree representing the interface, with a subdirectory for each Ethernet packet type. Figure 1 shows the files presented by a Plan 9 Ethernet driver.

Protocol stacks similarly serve a small directory tree. In normal use, a protocol stack is bound to an interface, in part by merging their served files in a *union directory* such as shown in Figure 2.

Figure 3 shows some representative contents of these files. This directory also contains files served by *dns*, the *connection server*[30], *cs*, and *ssl*.

Making an out-going connection is done by calling the *dial* function with an argument such as `tcp!collyer.net!ssh`. *Dial* communicates with *cs* by reading and writing `/net/cs`, if it exists. *Cs* in turn knows the various network media and protocols available, and how to translate names to numeric addresses, if necessary, and so places the call and tells *dial* what file to open in */net/tcp* (for example) to get a file descriptor for the resulting connection. *Cs* is a multi-process program, so it does not serialise its callers' access to the network, and thus isn't a bottleneck. Ordinary programs are thus isolated from the specifics of making network connections and adapt trivially to new media (e.g., ATM) and new transport protocols (e.g., IPV6, AAL5). To add support for new protocols or media, only *cs*, *dns*, *listen*, and the kernel need to be updated and rebuilt; ordinary programs will work with the new protocols and media unchanged. As a further benefit, shell scripts can make connections by dealing with the raw network plumbing (files under `/net`) directly.

Competitive testing has shown that the Plan 9 IP stack is the most robust one known. This stack supports IP V4 and V6 within a single stack, using IP V6 addressing internally.

4.6. Better Protocols

A distributed system lives and dies by the quality of its network protocols and their implementations. Internally Plan 9 uses these protocols: *9fs*, *exportfs*, *rexexec*, *ncpu*, *venti*. *9fs* runs over TCP and the obsolescent IL protocol, the others run over TCP only. All were designed specifically for Plan 9. *9fs* is 9P, the file system protocol that ties the whole distributed system together. It is used to communicate with device drivers and file servers alike.

Plan 9 also contains support for, or knowledge of, another 97 UDP or TCP Internet protocols. These Internet protocols are generally rather poorly designed and hobbled by self-imposed limitations. FTP, the

Copyright © 2003 Geoff Collyer

```
cpu% cd /net/ether0
cpu% ls -l *
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 0/ctl
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 0/data
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 0/ifstats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 0/stats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 0/type
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 1/ctl
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 1/data
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 1/ifstats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 1/stats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 1/type
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 2/ctl
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 2/data
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 2/ifstats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 2/stats
--r--r--r-- l 0 bootes bootes 0 May  9 15:42 2/type
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 addr
--rw-rw-rw- l 0 bootes bootes 0 May  9 15:42 clone
cpu% cat 0/stats
in: 7654260
out: 7334433
crc errs: 0
overflows: 0
soft overflows: 0
framing errs: 0
buffer errs: 0
output errs: 0
prom: 0
mbps: 100
addr: 00a0c91f4069
```

Figure 1: Plan 9 Ethernet driver's files

File Transfer Protocol, for instance, is excessively complex and awkward and doesn't provide access to much of the metadata in a modern file store. It has had to be modified to cope with the wide-spread use of firewalls (by reducing its complexity somewhat). SMTP, the Simple Mail Transfer Protocol, is not simple but its insistence on using only the low-order 7 bits of each 8-bit byte limits its capabilities and pushes limitations back into other mail standards, such as MIME. SMTP is a rather slow way to move mail, being a lock-step protocol that forces each party in the conversation to frequently wait for the other. Yet these protocols persist because of the extreme political difficulties in amending or replacing them. Many such protocols could be superseded by a common file system protocol with agreed-upon authentication, and 9P2000 (the latest revision of 9P) would be a good candidate.

The Internet protocols suffer from being largely designed in isolation. Thus we have a tower of Babel of authentication mechanisms and models. *factotum*(4) copes with the welter of incompatible external authentication mechanisms, thus providing the convenience of single sign-on, but the underlying mess is still there.

4.7. Archival Storage

Plan 9 file servers provide archival storage. *venti*(8) and *fossil*(4) together provide Plan 9 file service while conserving storage. The original Plan 9 file server, written by Ken Thompson, provides archival storage to optical media[31, 35]. Both file servers provide nightly snapshots which are saved forever, unlike those provided by various file server appliances, which vanish after a while. It turns out to be handy to have complete access to the file server's history. (One can run *fossil* using a *venti* store as its permanent storage, which in turn is stored on an optical-disc jukebox via Ken's file server, *fs*(8), thereby conserving optical-

Copyright © 2003 Geoff Collyer

```
cpu% cd /net
cpu% ls -l *
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 arp
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 bootp
--rw-rw-rw- M 41999 geoff   geoff  0 May 16 02:11 cs
--rw-rw-rw- M 42001 geoff   geoff  0 May 16 02:11 dns
d-r-xr-xr-x l     0 bootes  bootes 0 May  9 15:42 ether0/0
d-r-xr-xr-x l     0 bootes  bootes 0 May  9 15:42 ether0/1
d-r-xr-xr-x l     0 bootes  bootes 0 May  9 15:42 ether0/2
--rw-rw-rw- l     0 bootes  bootes 0 May  9 15:42 ether0/addr
--rw-rw-rw- l     0 bootes  bootes 0 May  9 15:42 ether0/clone
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 icmp/clone
--r--r--r-- I     0 network bootes 0 May  9 15:42 icmp/stats
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 icmpv6/clone
--r--r--r-- I     0 network bootes 0 May  9 15:42 icmpv6/stats
d-r-xr-xr-x I     0         bootes 0 May  9 15:42 il/0
d-r-xr-xr-x I     0 bootes  bootes 0 May  9 15:42 il/1
[ ... ]
d-r-xr-xr-x I     0 network bootes 0 May  9 15:42 il/9
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 il/clone
--r--r--r-- I     0 network bootes 0 May  9 15:42 il/stats
--r--r--r-- I     0 network bootes 0 May  9 15:42 ipgate6
d-r-xr-xr-x I     0         bootes 0 May  9 15:42 ipifc/0
--rw-rw-rw- I     0 network bootes 0 May  9 15:42 ipifc/clone
--r--r--r-- I     0 network bootes 0 May  9 15:42 ipifc/stats
--rw-rw-rw- I     0 network bootes  0 May  9 15:42 iproute
--r--r--r-- I     0 network bootes  0 May  9 15:42 ipselftab
--rw-rw-rw- I     0 network bootes  0 May  9 15:42 log
--rw-rw-rw- I     0 network bootes 60 May  9 15:42 ndb
d-r-xr-xr-x D     0 geoff   bootes  0 May  9 15:42 ssl/0
d-r-xr-xr-x D     0 bootes  bootes  0 May  9 15:42 ssl/1
d-r-xr-xr-x D     0 geoff   bootes  0 May  9 15:42 ssl/2
--r-xr-xr-x D     0 bootes  bootes  0 May  9 15:42 ssl/clone
d-r-xr-xr-x I     0 bootes  bootes  0 May  9 15:42 tcp/0
d-r-xr-xr-x I     0 bootes  bootes  0 May  9 15:42 tcp/1
[ ... ]
d-r-xr-xr-x I     0 network bootes  0 May  9 15:42 tcp/87
d-r-xr-xr-x I     0 none    bootes  0 May  9 15:42 tcp/9
--rw-rw-rw- I     0 network bootes  0 May  9 15:42 tcp/clone
--r--r--r-- I     0 network bootes  0 May  9 15:42 tcp/stats
d-r-xr-xr-x I     0 bootes  bootes  0 May  9 15:42 udp/0
d-r-xr-xr-x I     0 bootes  bootes  0 May  9 15:42 udp/1
[ ... ]
d-r-xr-xr-x I     0 network bootes  0 May  9 15:42 udp/4
--rw-rw-rw- I     0 network bootes  0 May  9 15:42 udp/clone
--r--r--r-- I     0 network bootes  0 May  9 15:42 udp/stats
```

Figure 2: Plan 9 network interface files

disc storage.) The nightly snapshots appear as complete file system trees under `/n/dump/2003/0513`, for example, for 13 May 2003, yet record only changed blocks in permanent storage. The ability to view and retrieve files from the past with ordinary commands such as *cp* and *cat* makes it safe to experiment, knowing that it's easy to restore files if the experiment fails.

```
cpu% cd /net
cpu% cat arp
ether  OK      10.9.0.3                                  00a0c91f4069
ether  OK      10.9.0.1                                  00a0c9e02756
ether  OK      10.9.0.4                                  0002e3028e78
ether  OK      10.9.0.6                                  000094d23e8a
ether  OK      10.9.0.10                                 00a0c9e0162c
cpu% cat iproute
10.0.0.0        /128 10.0.0.0        4b   ifc    -
10.9.0.0        /112 10.9.0.0        4i   ifc    0
10.9.0.0        /128 10.9.0.0        4b   ifc    -
10.9.0.3        /128 10.9.0.3        4u   ifc    0
10.9.255.255    /128 10.9.255.255    4b   ifc    -
10.255.255.255  /128 10.255.255.255  4b   ifc    -
255.255.255.255 /128 255.255.255.255 4b   ifc    -
```

Figure 3: Contents of Plan 9 network interface files

5. Programmers' View

5.1. Uniform File System Interface vs. Big Libraries and APIs

Functionality that might be provided by shared libraries or even conventional libraries on other systems is often provided on Plan 9 via file servers, whether in-kernel, such as provided by device drivers, or in user mode, as is more common. Some of these file servers are surprising to first-time Plan 9 users since they include the window system, *rio*(4), the dumps (backup system), `/proc`[19], which Linux is emulating (poorly), programs to interpret CDs, foreign filesystems and tapes in various old formats, a spam filter, *ratfs*(4), a program to present mailboxes as file trees, *upasfs*(4), the environment, *env*(3), the IP protocol stack, *ip*(3), a serial-console multiplexor, *consolefs*(4), and an FTP client, *ftpfs*(4).

These file servers all present their interfaces using the 9P remote file system protocol, which serves as the glue that ties the Plan 9 distributed system together. Unlike NFS, 9P can serve files with dynamic contents and yet permits caching. Since all file accesses in Plan 9 are through 9P, 9P has to be able to cope with any type of file, also unlike NFS. 9P makes it possible and efficient to provide many services outside the kernel but with a file system interface; a support library, *9p*(2), makes it relative painless to write such servers.

5.2. Processes and Threads

Plan 9 has one weight of process (light), but programs can control which of their resources are shared across a fork (see *rfork* in *fork*(2)). For those who like the added complexity of multiple non-concurrent threads sharing address space, there's a thread library that uses no kernel support to implement threads. Plan 9 has no *select* system call, which tends to force programs to become giant, unstructured event loops. Instead, programs use multiple processes or threads to wait for events to happen. Since processes are cheap, this is not a performance disaster.

5.3. Better Libraries

☞ **WARNING:** It's time to be brutally honest again. Take a deep breath. ☜ Plan 9's libraries were built from the beginning to support multiprocessing well, in contrast to the situation on UNIX, where libraries have been hammered and twisted to try to adapt to multiple processes or threads sharing address space, usually not very well (e.g., where is *fselect*, a *stdio* version of *select* that takes buffering into account?). Almost from the beginning, the libraries have been built to support the Unicode character set and its UTF-8 encoding as the native forms of each on Plan 9. Since ASCII is a proper subset of both, this is not as traumatic as a system which makes no attempt to encode Unicode characters (which doesn't really work in a world that has pipes and network connections) or which uses some warped encoding, such as

UTF-7. In particular, a stream of ASCII characters is also a valid UTF-8 representation of those characters, so ASCII files work unchanged in a UTF-8 world.

5.4. Better Compilers

Ken Thompson wrote the Plan 9 C compilers[36], assemblers and loaders. The compilers are fast, the loaders are smart and the assemblers are rarely used (ordinary C compilation bypasses them). All of these programs are cross-compilers, -assemblers and -loaders. All the compilers are related and compile the same language, so there's almost no need for conditional compilation. They can be run on Plan 9 systems of any architecture and generate code for any supported architecture. It's common, assisted by *mk*(1), to generate binaries of a program for all architectures at once, relying heavily on parallel compilation, again courtesy of *mk*. There is one header file per library and headers are included in a defined order, eliminating the clutter and inefficiency of UNIX's headers. The headers use `#pragma` to tell the loader which libraries to link against, further simplifying compilation.

5.5. Portability

Almost the entire system is highly portable. Obviously, parts of the kernel and the compilers, assemblers and loaders are machine-dependent, but once those have been written or adapted for a new architecture, the rest of the system just compiles and works, and this has been done repeatedly for new hardware[13].

5.6. Importing UNIX Programs

There is an ANSI/POSIX[5, 12] emulation environment (APE) used to import big UNIX programs that change frequently and thus aren't worth the effort to adapt to Plan 9 (repeatedly), such as *ghostscript*.

5.7. Efficiency

Plan 9 is efficient enough to run on small systems, though the definition of 'small' keeps growing. Machines large enough to run Microsoft Windows comfortably are always excessively large by Plan 9 standards, so Plan 9 continues to run well even on ''ancient'' hardware.

6. Security & Configurability

6.1. Namespaces

Plan 9 permits each process to have its own, distinct (file) namespace. More so than in UNIX, *everything* that can be addressed has a name in the file system namespace, which doesn't mean that the thing *is* a conventional disk file, merely that it looks like one. There are conventional names for things, such as `/bin` for the directory where executable programs and scripts are found. Union directories replace the many search path mechanisms of UNIX. Namespaces are malleable and provide simple solutions to multiplexing (e.g., in the window system) and permit substitution of resources, including those from other machines. No sound card on your machine? Import your neighbour's (with his consent) and use it as if it were your own. The malleability of namespaces is at odds with UNIX set-id, but Plan 9 doesn't have set-id; instead it has network services that perform cryptographic authentication of user requests.

6.2. Authentication

Secstore(1) provides secure encrypted storage of passwords and keys of all sorts, and *factotum*(4) implements single-sign-on and key management in general[9]. Plan 9 services do not exchange clear-text passwords among themselves over the network; occasionally Plan 9 programs provide options to permit clear-text passwords to cope with (broken) protocols as such as POP that encourage them. Plan 9 has no super-user. There are local host owners, with additional privileges over their particular machines, but those privileges do not extend over the network. Users and groups are unified and greater use of groups, and the willingness to trust groups and their administration, minimises the need for super-user access.

Copyright © 2003 Geoff Collyer

7. What’s Missing?

web browsers, use *vncv*(1) (Plan 9 is a distributed system) or *emu*(9).

Microsoft software

Drivers for random devices.

8. Summary

Plan 9 solves problems in distributed computing, file storage, internationalisation and system administration that still plague UNIX and other systems. It also provides a pleasant and powerful programming environment in which to tackle problems that it doesn’t solve right out of the box. It’s available for free and can be used commercially without fees or royalties. It runs on common platforms (PCs, Ipaqs) and some less common ones (Power PC evaluation boards), and is relatively easily ported to new hardware (the limiting factor here is usually obtaining adequate programming documentation for the hardware). In the past, it has run on NeXT, SGI and Sun machines.

A quarter-century ago, Doug McIlroy, Elliot Pinson and Berkeley Tague wrote in The Bell System Technical Journal, ‘‘UNIX is not the end of the road in operating systems innovations, [...]’’[22], but many people still seem to think that it is, and I include Linux as a UNIX clone. Even a system as well-designed and malleable as UNIX originally was eventually exceeds its design limitations, particularly in the hands of those who are not master programmers, as the creators of UNIX are and were. Attempts to delay facing that reality result in a bigger, clunkier system that becomes less elegant and efficient each day, without much substantive improvement[14].

Windows XP ... is the ‘‘most reliable Windows ever.’’ To me, this is like saying that asparagus is ‘‘the most articulate vegetable ever.’’ − Dave Barry

Microsoft’s Windows is not a direction that I want to go. Its graphics are prettier than UNIX’s, but the rest of the system, particularly the parts not visible on the screen, are complicated and have proven to be insecure. Without access to Windows source, it’s hard to know about bad the complexity and insecurity really are. The system does not seem elegant and I don’t see progress relative to UNIX here. Microsoft seems to prefer complex, ever-churning APIs to simple file system interfaces.

It’s time to move on. It has been for a while.

9. References and Further Reading

As always when dealing with the Internet, all RFCs cited may have been updated or replaced by later ones.

1. *Berkeley Software for UNIX on the VAX: 4.1bsd version of May, 1981,* Univ. of California at Berkeley (May 1981).
2. *System V Interface Definition,* AT&T (1985).
3. *Inferno 2.0 Reference Manual,* Lucent Technologies (1997).
4. Eric Allman and Miriam Amos, ‘‘Sendmail Revisited,’’ pp. 547-555 in *USENIX Conference Proceedings,* USENIX, Portland, OR (Summer 1985).
5. American National Standards Institute, X3J11 committee, *American National Standards Institute X3.159-1989 -- Programming Language C, = ISO/IEC 9899:1990,* ANSI, New York (1989).
6. James M. Bloom and Kevin J. Dunlap, ‘‘Experiences Implementing BIND, A Distributed Name Server for the DARPA Internet,’’ pp. 172-181 in *USENIX Conference Proceedings,* USENIX, Atlanta, GA (Summer 1986).
7. Computing Science Research Center, AT&T Bell Laboratories, Murray Hill, New Jersey, *UNIX Research System Programmer’s Manual, Tenth Edition,* Saunders College Publishing (1990).
8. The Unicode Consortium, *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volume 1,* Addison Wesley, New York (1991).
9. Russ Cox, Eric Grosse, Rob Pike, Dave Presotto, and Sean Quinlan, ‘‘Security in Plan 9,’’ in *Plan 9 Programmer’s Manual, Fourth Edition* (April 2002).

Copyright © 2003 Geoff Collyer

10. S. M. Dorward, R. Pike, D. M. Ritchie, H. W. Trickey, and P. Winterbottom, ‘‘Inferno,’’ *Proc. IEEE Computer Conference (COMPCON)*, San Jose, California (1997).

11. S. M. Dorward, R. Pike, and P. Winterbottom, ‘‘Programming in Limbo,’’ *Proc. IEEE Computer Conference (COMPCON)*, San Jose, California (1997).

12. Institute of Electrical and Electronics Engineers, *Portable Operating System Interface (POSIX), Part 1: System Application Program Interface (API) [C Language] (IEEE Std 1003.1-1990) = ISO/IEC 9945-1:1990*, IEEE, New York (1990).

13. Bob Flandrena, ‘‘Adding Application Support for a New Architecture in Plan 9,’’ in *Plan 9 Programmer’s Manual, Fourth Edition* (April 2002).

14. C. H. Forsyth, ‘‘More Taste: Less Greed? or, Sending UNIX to the Fat Farm,’’ pp. 161−172 in *Proceedings of the Summer 1990 UKUUG Conference*, UKUUG, London (July 9−13, 1990). `http://www.caldo.demon.co.uk/doc/taste.pdf`

15. James Gosling and William N. Joy, *The Java Programming Language*, Sun Microsystems.

16. C. A. R. Hoare, ‘‘Communicating Sequential Processes,’’ *Communications of the Association for Computing Machinery (ACM)* **21**(8), pp. 666-677 (1978).

17. K. Jensen, *Pascal User Manual and Report*, Springer-Verlag (1978). (2nd edition)

18. B. W. Kernighan, ‘‘Why Pascal is Not My Favorite Programming Language,’’ Comp. Sci. Tech. Rep. No. 100 (July 1981).

19. T. J. Killian, ‘‘Processes as Files,’’ *USENIX Association Conference Proceedings*, Salt Lake City, Utah, pp. 203-207 (Summer 1984).

20. AT&T Bell Laboratories, in *UNIX Programmer’s Manual, Ninth Edition, Volume One*, ed. M. D. McIlroy, Murray Hill, New Jersey (September 1986).

21. Bell Laboratories, *UNIX Programmer’s Manual*, Holt, Rinehart and Winston (1982).

22. M. D. McIlroy, E. N. Pinson, and B. A. Tague, ‘‘UNIX Time-Sharing System: Foreword,’’ *Bell Sys. Tech. J.* **57**(6), pp. 1899-1904 (1978).

23. M. D. McIlroy, ‘‘A Research UNIX Reader: Annotated Excerpts from the Programmer’s Manual, 1971-1986,’’ Comp. Sci. Tech. Rep. No. 139 (June 1987).

24. Rob Pike, ‘‘Window Systems Should Be Transparent,’’ pp. 279-296 in *Computing Systems*, USENIX (Summer 1988).

25. Rob Pike, ‘‘A Concurrent Window System,’’ pp. 133-153 in *Computing Systems*, USENIX Association (Spring 1989).

26. Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, Tom Duff, and Gerard Holzmann, ‘‘Plan 9: The Early Papers,’’ Comp. Sci. Tech. Rep. No. 158 (July 1991).

27. Rob Pike, ‘‘8-1/2, the Plan 9 Window System,’’ pp. 257-266 in *USENIX Conference Proceedings*, USENIX, Nashville, TN (Summer 1991). FTP - research.att.com:/dist/plan9doc/4; local - 8half.ps

28. Rob Pike, ‘‘Acme: A User Interface for Programmers,’’ pp. 223-234 in *USENIX Conference Proceedings*, USENIX, San Francisco, CA (Winter 1994).

29. David L. Presotto, ‘‘Upas - a simpler approach to network mail,’’ pp. 533-538 in *USENIX Conference Proceedings*, USENIX, Portland, OR (Summer 1985).

30. D. L. Presotto and D. M. Ritchie, ‘‘Interprocess Communication in the Eighth Edition Unix System,’’ pp. 309−316 in *Usenix Summer Conference Proceedings, Portland 1985*, Usenix, Portland, OR, USA (June 11−14, 1985).

31. Sean Quinlan, ‘‘A Cached WORM File System,’’ *Software—Practice and Experience* **21**(12), pp. 1289−1299 (December 1991).

32. D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, ‘‘UNIX Time-Sharing System: The C Programming Language,’’ *Bell Sys. Tech. J.* **57**(6), pp. 1991-2019 (1978).

Copyright © 2003 Geoff Collyer

33. D. M. Ritchie and K. Thompson, ‘‘The UNIX Time-Sharing System,’’ *Bell Sys. Tech. J.* **57**(6), pp. 1905-1929 (1978).

34. Lucent Technologies, ‘‘The Limbo Language Definition,’’ in *Inferno User’s Guide* (1997).

35. Ken Thompson, ‘‘The Plan 9 File Server,’’ in *Plan 9 Programmer’s Manual, Fourth Edition* (April 2002).

36. Ken Thompson, ‘‘Plan 9 C Compilers,’’ in *Plan 9 Programmer’s Manual, Fourth Edition* (April 2002).

37. C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M. Crispin, and P. Svanberg, *RFC 2130: The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996,* ISI, Marina Del Ray, CA (April 1997).

38. F. Yergeau, *RFC 2279: UTF-8, a transformation format of ISO 10646,* ISI, Marina Del Ray, CA (January 1998).

Copyright © 2003 Geoff Collyer