The Unix Spirit set Free: Plan 9 from Bell Labs

Uriel

uriel@cat-v.org

The most common question about Plan 9

Is Plan 9 Free Software/Open Source?

Yes!

Latest Plan 9 license is considered Free Software by RMS and the FSF.
Considered Open Source by the OSI.
Free according to the DFSG.

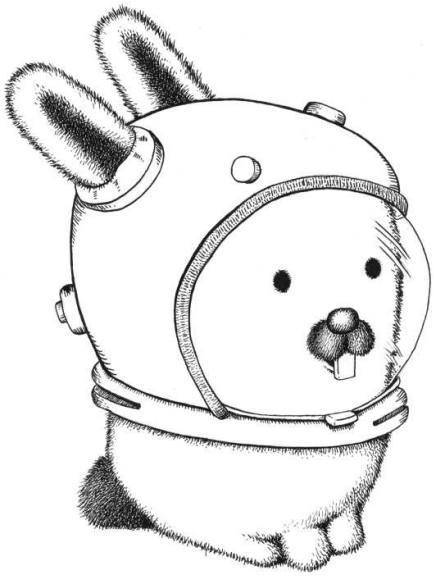
With politics out of the way, lets get on with the important stuff...

The JWZ test

Your "use case" should be, there's a 22 year old college student living in the dorms. How will this software get him laid?

– Jamie Zawinski

Glenda, the Plan 9 Bunny



The cutest mascot

The history of Plan 9

Started in the mid 1980's at Bell Labs by the same team that created Unix.
First full time users in 1989, in production since then.

Four public releases

1st Ed (1993): First public release, only to universities.
2nd Ed (1995): First general public release.
Inferno 1st Ed (1997): Based in many of the ideas and technologies in Plan 9.
3rd Ed (2000): First free release including source code; codename *Brazil*.
4th Ed (2002): Approved Open Source license.
Inferno 4th Ed (2005): Released under GPL/MIT license.

Still in active development

Daily ISO builds.
Continuous updates delivered over 9p/replica.
Any user can submit changes using a simple patch creation and tracking system.
Repository of contributed programs and other resources in */n/sources/contrib/*.

Some background history: In the beginning...

Multics

- Multiuser.
- Multiprocessing.
- Multiprogramming.
- Big, slow, bloated and ***complex***.

While Multics collapsed under its own weight, Ken started to play with a PDP-7.

Unix

What was different: Small, fast, lean and ***simple***.

In other words: KISS

Unix distilled and polished many of the Multics ideas, and added some of its own.

Hierarchical file system.
Everything is a file.
Each tool does one thing and does it well.
Processes communicate and interact through pipes.
Text is the universal language.
Written in a high-level language.

This was mostly true in the 1970's, since then, little has improved and some things now are worse.

What was wrong with Unix?

"Not only is UNIX dead, it's starting to smell really bad."
– Rob Pike circa 1991

Designed as an old fashion timesharing system, has trouble adapting to a world of networks and workstations.

The advantages of timesharing were lost in the switch to workstations: Centralized management and administration, amortization of costs and resources.

Many features badly retrofitted over the years (eg., graphics, networking.)

Lots of hanging historical baggage.

Loss of conceptual integrity.

Unix is not ***simple*** anymore.

What Plan 9 doesn’t have...

root, suid
tty, curses
ioctl
sockets
select/poll
symlinks
pthreads
mmap
dynamic linking
loadable kernel modules
locales
gcc
C++
emacs (or vi)
X11
XML
WEB 2.0

Plan 9 is a completely new system

Plan 9 distills and polishes many of the Unix ideas; and adds some of its own.

New kernel: designed to be portable and support SMP.

New compilers: small, fast and portable, cross compiling is the same as compiling.

New libraries: simpler and less error-prone.

New user interface: takes advantage of modern display and input devices.

New tools: debugger, text editors, mail system, network database, etc.

All components designed to work well together in a distributed environment.

Back to the Unix roots: Simplicity, Clarity and Generality

Note: To really understand Plan 9, you have to ***unlearn*** the last 30 years of Unix. The similarities are big enough to create false expectations.

Main ideas

Three major concepts

Everything is a file tree (*file system*): ***All*** resources are named and accessed like files in a hierarchical file system. No files are more special than others.

9P: A single protocol to securely and transparently access resources independently of their location.

Private namespaces: Every process can customize its view of the world by changing its private namespace that joins together the file hierarchies provided by different servers representing the network resources.

All resources as file systems

Networking: */net/*
Authentication: */mnt/factotum/*
Encryption: */net/tls/*
Graphics: */dev/draw/*
Window system: */dev/wsys/*
Process control and debugging: */proc/*
Environment: */env/*
Email: */mail/fs/*
Boring stuff: cdfs, webfs, tarfs, ftpfs, etc...
Weird stuff: wikifs, gpsfs, sshnet, iostats and others.

The 9P protocol

Lightweight network file system (but in Plan 9, *everything* is a file.)

- Not block oriented, byte oriented.
- Minimalistic and lightweight: Only 13 message types (NFSv3 has 22, NFSv4 has 42)
- Can run over any reliable transport (directly over TCP, IL, RUDP, PPP, shared memory, serial port connections, PCI bus and others)
- Encompassing: used for local and remote access; for *synthetic* and *physical* files.
- Authentication and encryption agnostic.
- Designed for transparent caching and stacking.

Namespaces

Imagine using a programming language where all variables were global...

...that is how Plan 9 users feel when they have to use Unix.

Three namespace operations:

```
int bind(char *name, char *old, int flag)

int mount(int fd, int afd, char *old, int flag,char *aname)

int unmount(char *name, char *old)
```

Example: */sys/src/libdraw/newwindow.c*

From the shell:

```
% mount `{cat /env/wsys} /n/foo new
% cat /proc/$pid/ns
```

Union mounts

Flags for *bind* and *mount*:

MREPL: Replace the old file by the new one. Henceforth, an evaluation of old will be translated to the new file.

MBEFORE: Both the old and new files must be directories. Add the files of the new directory to the union directory at old so its contents appear first in the union.

MAFTER: Like MBEFORE but the new directory goes at the end of the union.

MCREATE: OR'd with any of the above. When *create* attempts to create in a new file in a union directory will create the file in the first mount with the *MCREATE* flag.

Example:

```
% ns | grep /bin
```

The kernel

Linux

Over 300 syscalls and counting; hundreds (thousands?) of ioctls.

4,827,671 lines of code.

Others not better... Unix not small anymore.

Plan 9

37 syscalls; no networking related syscalls; no ioctls.

148,787 lines of code.

Optional real-time scheduler.

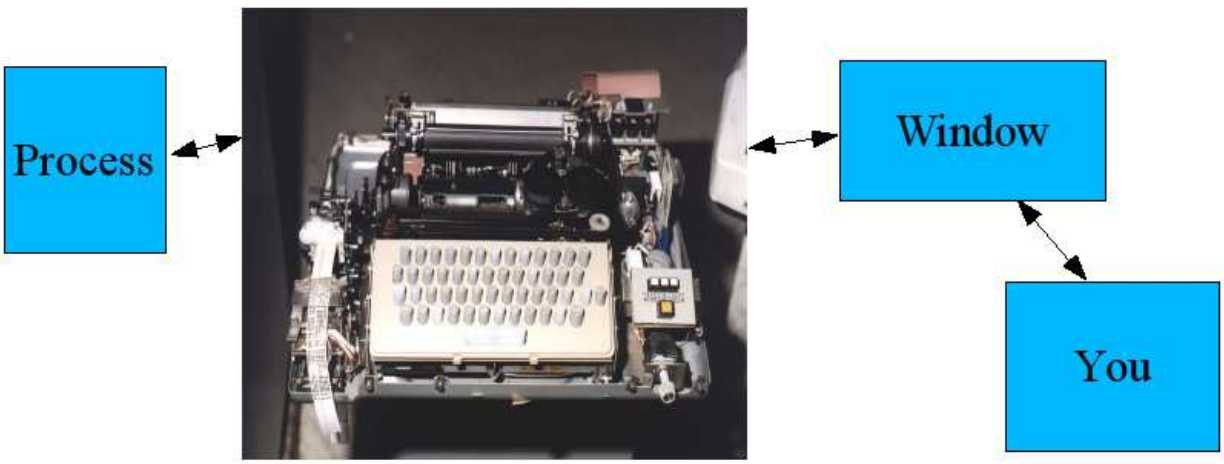
Microkernel or monolithic kernel? Who cares?

Inside the kernel or outside the kernel? Do what makes most sense, can change it later without breaking the interface.

Pragmatic, not dogmatic design.

Unix: X terminal (emulator)

Wow, how advanced; 1,000,000 times the performance and there's still an ASR-33 in the middle!



Don't believe me? Type 'stty' in an xterm and tell me why a window has a baud rate.

Rio - the Plan 9 window system

The Plan 9 user interface was designed to take full advantage of modern bitmapped displays and input mechanisms.

Yes, you really want to use a 3 button mouse.

All text is editable.

No cursor addressing.

Works as simple text editor.

Use the output of previous commands as the basis of new commands.

All applications transparently take advantage of its features: Plumbing, chording, hold mode, tab completion.

Multiplexes its environment: rio runs inside rio.

Who needs shell history when you can grep */dev/text*, edit the output, and click "Send"? Using the mouse *is* faster and much more *expressive* if the user interface designed to take advantage of it.)

Acme: a user interface for programmers

Windows organized in stacks. Makes management of many windows easy.

All text is editable and executable.

Mouse chording.

File system interface provides programability.

Acme applications: Wiki browser/editor, email and news clients, debugger interface, ...

Plumbing

Text everywhere can be plumbed.
Plumbing rules determine what to do.
Flexible and powerful.
File system interface.
Works across the network.

The cross compilers

One program per architecture.

Cross compiling is the same as compiling.

Very fast, I/O bound.

Reliable.

Very portable: aprox 7,000 lines of code per arch (gcc is >150,000 lines per arch.)

Existing ports to: Motorola MC68000/MC68020, StrongARM, Alpha, i386, amd64, SPARC, SPARC64, PowerPC, MIPS and others.

ken's C

Very simplified preprocessor.

Support for UTF-8 literals.

Some small and convenient extensions.

The acid debugging language

Implemented as a language-agnostic language to prove and manipulate processes.

Can write your own library of helpful functions in the acid language for specific tasks.

Interfaces with processes over /proc/, allows transparent remote debugging, even across different architectures!

Can take a *snap* of a process directly from /proc/ for later analysis. No need for core dumps!

Opening a network connection in Unix

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
    ...
struct sockaddr_in sock_in;
struct servent *sp;
struct hostent *host;
    ...
    memset(&sock_in, 0, sizeof (sock_in));
    sock_in.sin_family = AF_INET;
    f = socket(AF_INET, SOCK_STREAM, 0);
    if (f < 0)
         error("socket");
   if (bind(f, (struct sockaddr*)&sock_in, sizeof sock_in) < 0){
         error("bind");
    host = gethostbyname(argv[1]);
    if(host){
         sock_in.sin_family = host->h_addrtype;
         memmove(&sock_in.sin_addr, host->h_addr, host->h_length);
```

...

```
}else{
     sock_in.sin_family = AF_INET;
     sock_in.sin_addr.s_addr = inet_addr(argv[1]);
     if (sock_in.sin_addr.s_addr == -1)
         error("unknown host %s", argv[1]);
}
sp = getservbyname("discard", "tcp");
if (sp)
     sock_in.sin_port = sp->s_port;
else
     sock_in.sin_port = htons(9);
if (connect(f, (struct sockaddr*)&sock_in, sizeof sock_in) < 0){
     error("connect:");
```

Tedious and clunky
Protocol-specific details leak thru the API
C-specific, hard to access from elsewhere without ugly wrapping

Opening a network connection In Plan 9

```
#include <u.h>
#include <libc.h>
    ...
    fd = dial(netmkaddr(argv[1], "tcp", "discard"), 0, 0, 0);
    if(fd < 0)
         sysfatal("can’t dial %s: %r", argv[1]);
```

Clean and simple

Abstracts the protocol details from the application, addresses

/net

```
/net/cs
/net/dns
/net/tcp/
/net/udp/
/net/etherX/
```

No need for NAT or VPN, just import */net* from the g ateway or remote host, all applications will use it transparently.

Can mount multiple local IP stacks concurrently. Each stack is physically independent of all others. Normally a system uses only one stack. Multiple stacks can be used for debugging, implementing firewalls or proxy services, eg., ipmux.

Fossil and Venti

Venti provides block storage indexed by hash. Duplicated blocks stored only once.

Fossil: Permanent storage of files with automatic snapshots using Venti for storage.

Examples:

To see what changes have been made to the user database since the system was setup:

```
% history -D /adm/users
```

To compile a kernel with the version of the Intel PRO/1000 ethernet driver from 3 days ago:

```
% cd /sys/src/9/pc
% yesterday -b -n 3 ./etherigbe.c
```

Security, factotum and secstore

No root and no suid.

When everyone has the same privileges(none), escalation is impossible.

Namespaces provide isolation, much cleaner and secure than chroot.

Authentication and encryption implemented in a single place, thru file systems!

Factotum

Handles all authentication tasks.

Transparently for the application.

Manages keys, keeps private keys secret from applications!

Can store all keys securely in a *secstore* encrypted with a master key (often in remobable media).

Concurrency with Communicating Sequential Processes

Introduced by Hoare's 1978 paper, based on Dijkstra's work. Book online at:
http://www.usingcsp.com/

Procs: preemptively scheduled by operating system, shared address space.

Threads: cooperatively scheduled coroutines within a proc. (***Not*** pthreads! think coroutines.)

Channels: all communication is done through synchronous typed channels.

Avoids locking and simplifies design of concurrent systems.

Implementations:

Alef: compiled language, for systems.

Limbo: portable, bytecode-based. The main language used in Inferno.

Libthread: translate Alef programs to C. Lose syntactic sugar.

Others: Occam, concurrent ML, Moby, Erlang, StacklessPython.

Inferno/Limbo

Based on the ideas researched in Plan 9 but taking a more radical approach.

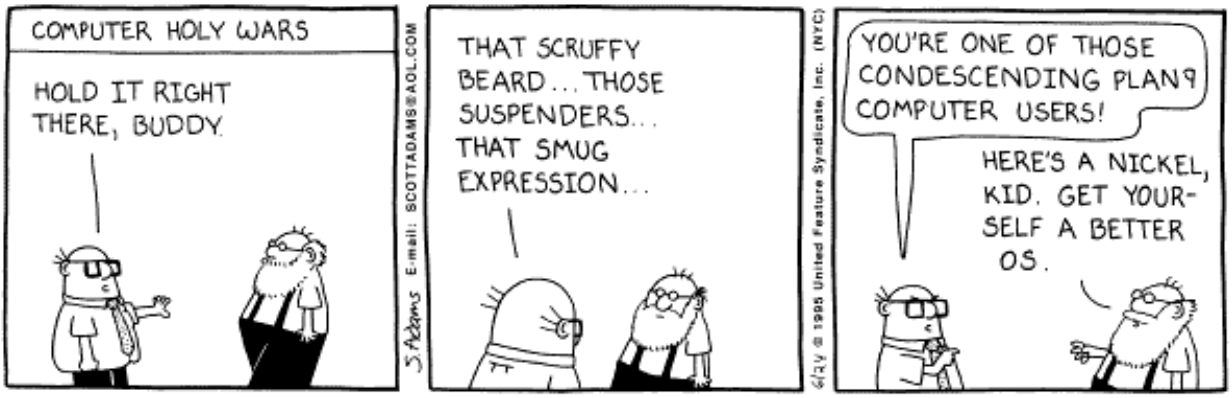
- Limbo: New GC concurrent language while keeping the C philosophy.
- Dis: Virtual machine designed for portability and JIT.
- Can run directly on hardware or hosted on almost any operating system (Plan 9, Linux, *BSD, Solaris, Irix, Windows, MacOS and others.)
- Extremely portable, doesn't require MMU!
- Processes **extremely** cheap.

Inferno and Plan 9 complement each other, for example:

- Can mount */net* in Plan 9 from Inferno instances running on any OS.
- Can debug embeded Inferno systems remotely from Plan 9.
- Can write cross platform applications in Limbo.
- Allows integration of non-Plan 9 systems in a Plan 9 network, Inferno Grid.

Released last year under a GPL/MIT dual license.

The Plan 9 community



If you thought BSD users were arrogant, you aint seen nuthing.

Conclusions

Plan 9 shares the Unix spirt of simplicity, clarity and generality.

But despite some superficial similarities, it is very different; be ready to unlearn all you know.

If you can’t give up what you are used to, there is still a lot to learn from Plan 9.

Chances are that the kind of problems you are facing are not too different from the ones Plan 9 has tried to solve; looking at Plan 9 you might be surprised by simpler solutions than you thought possible.

Resources and references

http://9fans.net - The main Plan 9 website.

http://plan9.bell-labs.com/sys/doc/ - The main collection of papers.

http://plan9.bell-labs.com/wiki/plan9/Papers/ - Other papers and publications.

http://www.vitanuova.com/inferno/net_download4T.html - Inferno.

http://plan9.us - Plan 9 from User Space for Unix systems.

http://www.usingcsp.com/ - *Communicating Sequential Processes* by Tony Hoare.