Multics, UNIX and Plan9

Nils Asmussen

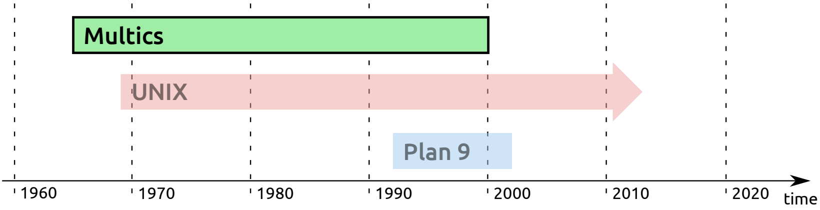
Influencial OS Research, 04/29/2014

Outline

Why is it part of IOS?

- We are still using the concepts of Multics/UNIX today
- Many if not most OSs today are UNIX-like (Linux, *BSD, Solaris, Mac OS, Minix, . . .)
- There has been a lot of research based on them
- And it's a big success in industry

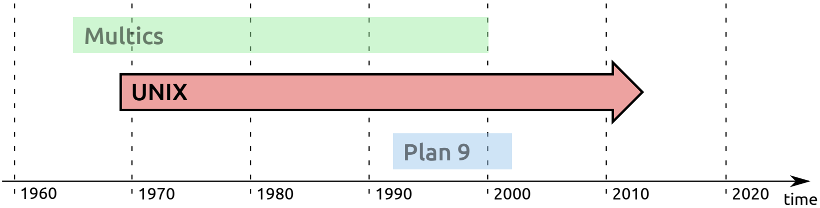
A brief overview



Multics

- Multics = Multiplexed Information and Computing Service
- Implemented in PL/I
- Last machine running Multics was shutdown in 2000

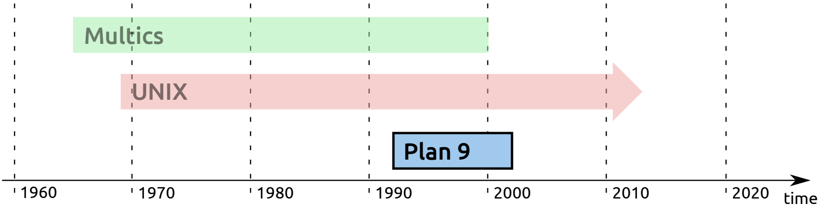
A brief overview



UNIX

- UNIX = UNiplexed Information and Computing Service
- Initially written in assembly, later rewritten in C
- Last UNIX from Bell Labs end of 80's; lot of derivatives

A brief overview

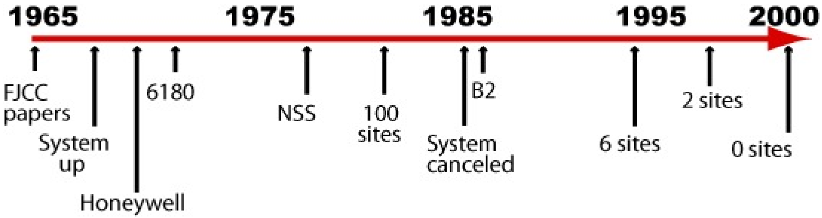


Plan 9 from Bell Labs

- Reference to the movie "Plan 9 from Outer Space"
- Implemented in C; temporarily in Alef
- First released 1992; commercial version in 1995; Open Source release in 2002

Outline

Timeline of Multics



References for Multics

1. **Structure of the Multics Supervisor**, 1965
   V. A. Vyssotsky; Bell Telephone Laboratories
   F. J. Corbató, R. M. Graham; MIT, Cambridge, Massachusetts
2. **A General-purpose File System for Secondary Storage**, 1965
   R. C. Daley; MIT, Cambridge, Massachusetts
   P. G. Neumann; Bell Telephone Laboratories
3. **The Multics Virtual Memory: Concepts and Design**, 1969
   A. Bensoussan, C.T. Clingen; Honeywell, Inc.
   R.C. Daley; MIT
4. **The Multics Input/Output System**, 1972
   R. J. Feiertag; MIT, Cambridge, Massachusetts
   E. I. Organick; University of Utah, Salt Lake City, Utah
5. **Official Website of Multics**
   `http://www.multicians.org`

Goals of Multics

- Allow changes and extensions
- Remove the boundary between OS and user applications
  - There has been a (soft) boundary before
  - But users try *really* hard to get around it
  - Why not remove it then?
  - OS can be changed like user apps (no special tools, . . .)
- Most users have no interest in computers and programming
  - Provide packages and frameworks that make it easy
- Security is important
  - Prevent unauthorized access to data
  - Hierarchical filesystem with protection mechanisms

Execution and Processes

- Process = a program in execution
- Has its own address space, based on segmentation and paging
- Processes are spawned from other processes. It can be specified what segments should be shared and what should be copied.
- OS and user applications use the same calling conventions
- OS sements/pages can be swapped out, too
- OS segments are shared among all processes
- Ring protection to prevent unauthorized access of OS segments

Scheduling

- Multics was designed for multiple CPUs
- Uses time-shared multiprogramming
- Has to cope with overload situations
  - Service denial or service degradation
  - Better minimize context-switches
  - Urgent jobs first, i.e. jobs where someone loses time and money not having the results
  - Urgency should be determined be humans
- It's name was "traffic controller"

Segmentation: Motivation

- Think of a segment as a segment in x86: a part of an executing program, together with meta-data like its length and access permissions.
- The authors desired to share information easily and in a controlled way
- Segmentation prevents that one needs explicit I/O calls to access the information
- One can just access it and the operating system makes sure that it is loaded from secondary storage or written back, if necessary.
- Additionally, sharing can be controlled, because the HW provides means to notice whenever a segment is used (and not loaded yet).

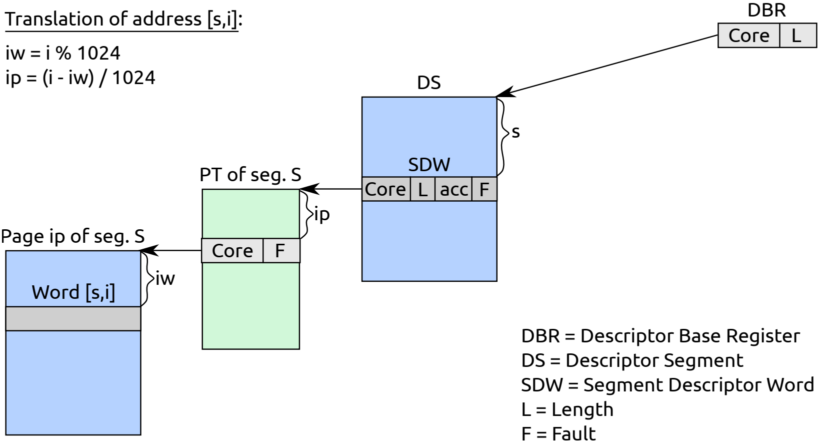
Paging: Motivation

- Swapping entire segments in and out is not feasible if segments are larger
- Fragmentation: growing and shrinking requires data-movement if the segment memory has to be contiguous
- Using variable sized pages complicates the management
- Thus, they split segments in fixed-sized pages

Segmentation and Paging

Translation of address [s,i]:

iw = i % 1024
ip = (i - iw) / 1024



DBR = Descriptor Base Register
DS = Descriptor Segment
SDW = Segment Descriptor Word
L = Length
F = Fault

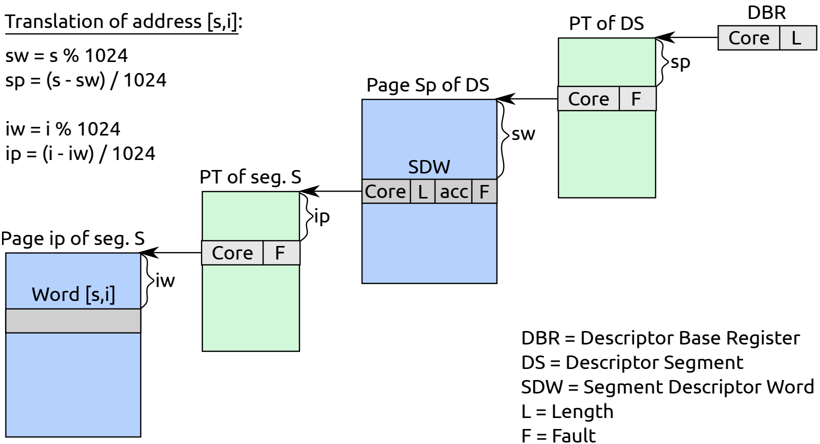
Figure: Hardware segmentation and paging in the Honeywell 645

Segmentation and Paging (with Paged DS)

Translation of address [s,i]:

sw = s % 1024
sp = (s - sw) / 1024

iw = i % 1024
ip = (i - iw) / 1024



DBR = Descriptor Base Register
DS = Descriptor Segment
SDW = Segment Descriptor Word
L = Length
F = Fault

Figure: Hardware segmentation and paging in the Honeywell 645

Segment Management

- Multics maintains a per-process table for segments that maps names to numbers
- The SDW is not set immediately, but on demand
- Pages in core are multiplexed among pages in virtual memory
- Selection algorithm based upon page usage (LRU); HW provides used-bit
- The OS decides which parts of a program lies in core where and when
- Exception: real-time $\rightarrow$ routines for:
  - Certain parts have to be in core
  - Certain parts are required soon
  - Certain parts won't be accessed again

Segment Hierarchy / Filesystem

- Multics invented hierarchical filesystems
- FS doesn't know about the format of files; only the user does
- Directory = special file with a list of entries maintained by the FS
- A directory entry may point to a file ("branch") or to another entry ("link")
- Branch contains physical address of the file, access time, permissions, ...
- They used a different notation:
  - "O" = root, not specified in paths
  - Absolute and relative paths: "A:B:C", ":*:*:B"

I've also seen "ROOT > A > B > C".

Dynamic Linking

- Works basically like today
- Procedure and data segments may contain unresolved references
- Procedure segments can be shared
  - Procedure segment is not changed
  - But has a linkage segment with entries consisting of:
    - Symbolic name of the externally known symbol
    - Symbolic name of the foreign segment
    - An indirect word; initially with a tag to cause a trap
- If not resolved yet, a trap will occur and the linker resolves it
- Allows to call segments of other processes

Input/Output System

- Had two main goals:
  - Simple things should be simple.
  - Complex things should be possible.

  (from Alan Kay)
- It should be device independent, as far as possible
  - Simplicity for the programmer
  - Less maintainance costs
  - Apps can use devices that the programmer didn't even think of

I/O System: Overview

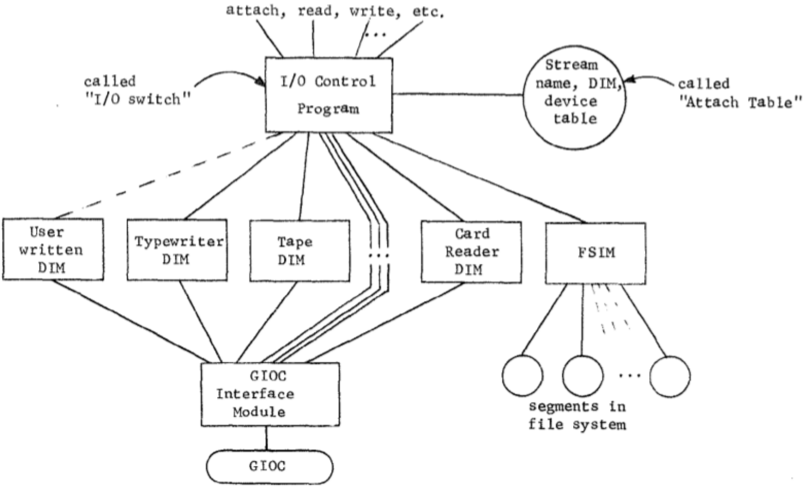
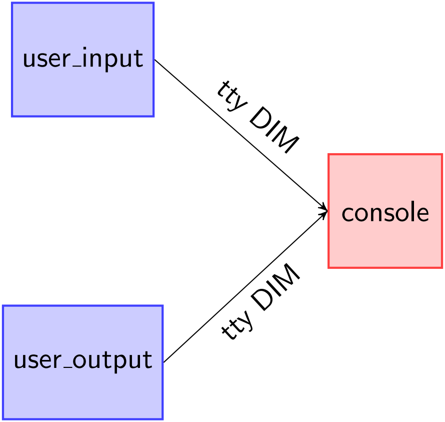


Figure 2 - Simplified view of I/O System organization.

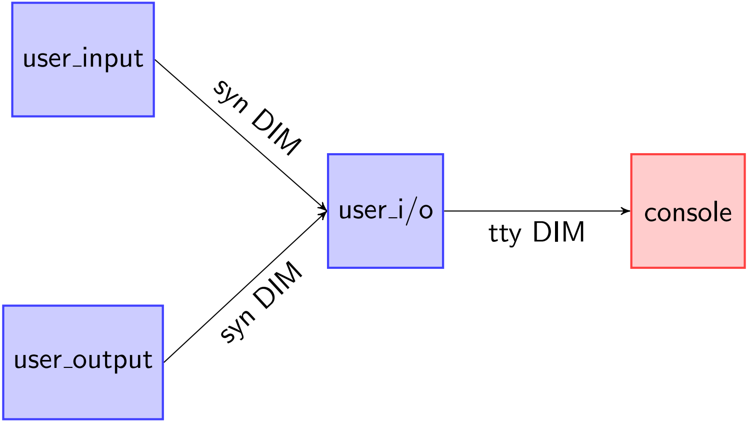
I/O System: Operations

- Init/deinit: `attach`, `detach`
- Positioning: `seek`, `tell`
- Read/write: `read`, `write`
- Read-ahead/write-behind: `readsync`, `writesync`, `resetread`, `resetwrite`
- Workspace (a)synchronous mode: `worksync`, `upstate`, `iowait`, `abort`
- Catch-all: `order`

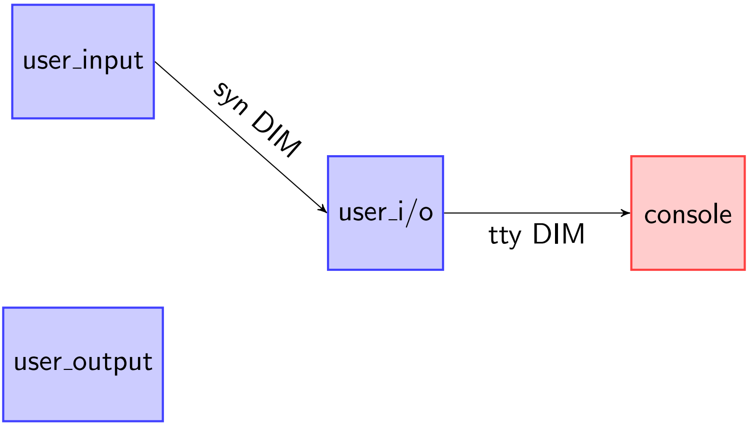
I/O System: Synonym Module



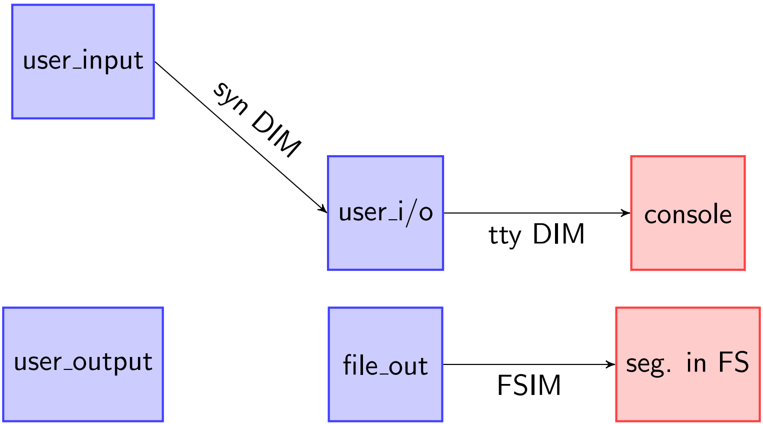
I/O System: Synonym Module



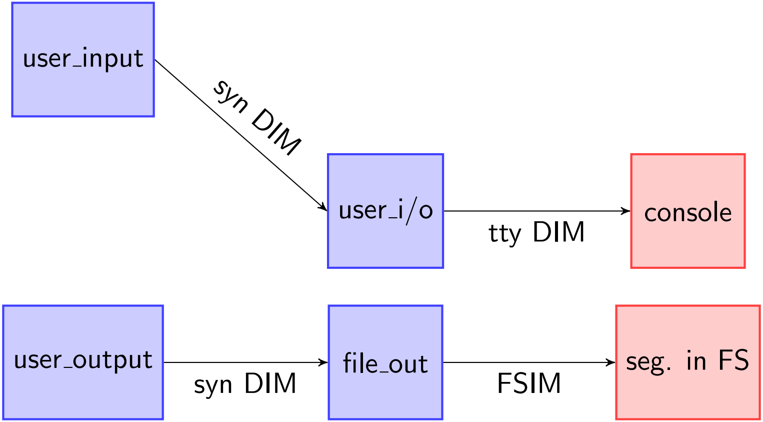
I/O System: Synonym Module



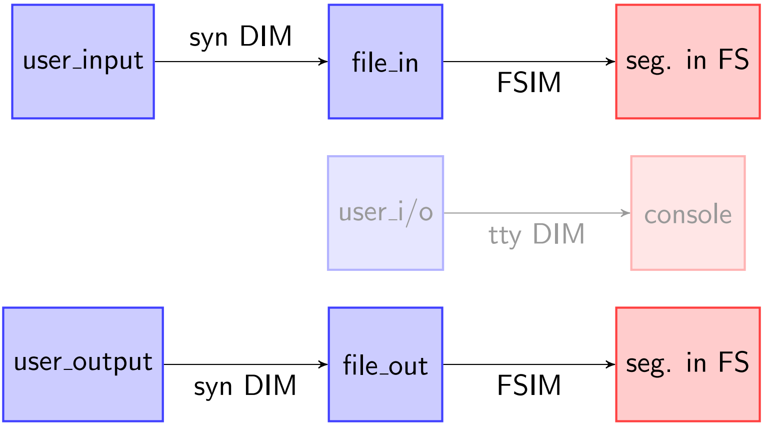
I/O System: Synonym Module



I/O System: Synonym Module

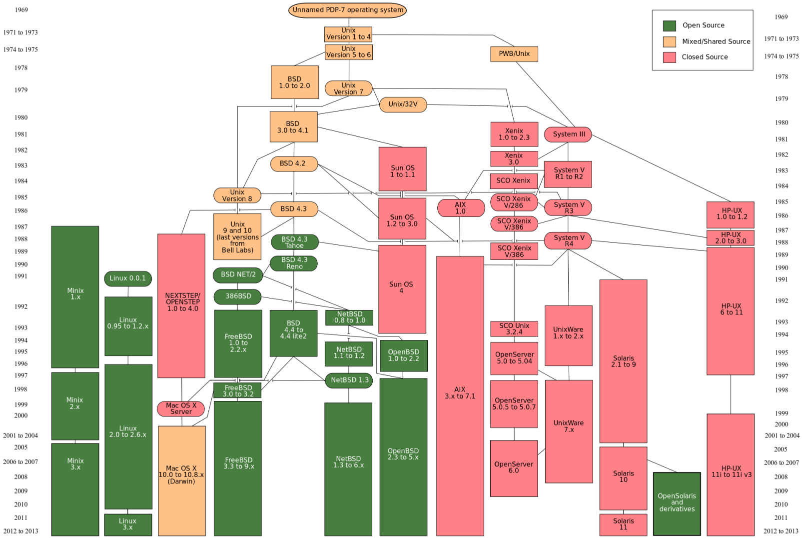


I/O System: Absentee process



Outline

Timeline


Open Source
Mixed/Shared Source
Closed Source
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
PWB/Unix
BSD 1.0 to 2.0
Unix Version 7
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
Sun OS 1 to 1.1
System V R1 to R2
SCO Xenix
Unix Version 8
SCO Xenix V/286
System V R3
AIX 1.0
BSD 4.3
Sun OS 1.2 to 3.0
SCO Xenix V/386
HP-UX 1.0 to 1.2
Unix 9 and 10 (last versions from Bell Labs)
BSD 4.3 Tahoe
System V R4
HP-UX 2.0 to 3.0
SCO Xenix V/386
BSD 4.3 Reno
BSD NET/2
Linux 0.0.1
Minix 1.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Sun OS 4
386BSD
NetBSD 0.8 to 1.0
HP-UX 6 to 11
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4 to 4.4 lite2
SCO Unix 3.2.4
UnixWare 1.x to 2.x
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
Solaris 2.1 to 9
NetBSD 1.3
Minix 2.x
FreeBSD 3.0 to 3.2
AIX 3.x to 7.1
Mac OS X Server
OpenServer 5.0.5 to 5.0.7
Linux 2.0 to 2.6.x
UnixWare 7.x
Minix 3.x
Mac OS X 10.0 to 10.8.x (Darwin)
FreeBSD 3.3 to 9.x
NetBSD 1.3 to 6.x
OpenBSD 2.3 to 5.x
OpenServer 6.0
Solaris 10
OpenSolaris and derivatives
HP-UX 11i to 11i v3
Linux 3.x
Solaris 11
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2013

References for UNIX

1. **The UNIX Time-Sharing System**, 1973
   Dennis M. Ritchie, Ken Thompson; Bell Laboratories
2. **The Evolution of the Unix Time-sharing System**, 1979
   Dennis M. Ritchie; Bell Laboratories

Motivation / origin story

- K. Thompson and D. Ritchie were working on Multics, but there was no usable system in sight
- During 1969, they seeked for an alternative
- Thompson and Ritchie started to design the filesystem on blackboards
- Thompson also created a fairly detailed performance simulation of the filesystem and the paging behaviour
- Thompson wrote a game called Space Travel for the GE-645, but CPU time was expensive
- Soon he found an unused PDP-7 and ported the game to it
- Building and deploying was quite tedious
  - He started with an OS for the PDP-7
  - Started with the blackboard-filesystem and processes
  - Finally an assembler and utilities to be self-hosted

Filesystem

- Most important aspect of UNIX ("everything is a file")
- In contrast to Multics, limited filename lengths (14 chars)
- Different syntax: "/" as separator, "/" = root
- Different semantics for links: there is no original
- A directory has at least the entries "." and ".."
- Special files for devices: device type + subdevice number
- At first: no path names, just file names; no dir creation at runtime
- New concept: mounting
- No links between different filesystems for bookkeeping reasons
- ACL-permissions with set-uid but without groups

I/O

- Like in Multics, a general interface for all devices and files
  - `open`, `read`, `write`, `close`, ...
- File descriptor
- Was word-based at the beginning, null-byte for padding
- No user-visible locks
  1. Not necessary: they were not faced with large files maintained by independent processes
  2. Not sufficient: can't prevent confusion (e.g. 2 users edit a copy of the same file in an editor)

Processes and Images

- "image" = computer execution environment (core image, registers, open files, cwd, ...)
- A process is the execution of an image
- User-part of core image consists of text (shared, read-only), data and stack
- `pid = fork(label)` (borrowed from the Berkeley time-sharing system)
- `execute(file, arg1, ..., argn)`
- `wait` and `exit`
- Pipes for IPC
- At the beginning: no multi-programming – switch was a complete swap

Synchronous IPC

- An early version had a similar primitive as sync. IPC
- Sender was blocked until receiver was ready
- Usages:
  - Instead of `wait` – parent did a send which returned an error if child exited
  - Init did a receive from every shell it created; on exit shell sended a message
- Was replaced with the less general mechanism `wait`

Init and shell

Initialization

- Done by init which forks a process for every typewriter
- Each waits for the user to login
- After login, it changes cwd, sets uid and exec's the shell
- Original init waits until a process died and restarts it

Shell

- If a command is not found, /bin/ is prefixed (no $PATH?)
- Standard streams: no stderr
- I/O redirection
- Filtering via pipes
- Background jobs

Traps and Signals

- The PDP-11 detects several HW faults and raises a trap
- Typically, the process is killed
- One can also send the interrupt signal to a process via the "delete" character
- The quit signal kills a process and produces a core image
- All signals can be ignored or handled

Outline

References for Plan9

1. **Plan 9, A Distributed System**, 1991
   Dave Presotto, Rob Pike, Ken Thompson, Howard Trickey
   AT&T Bell Laboratories
2. **Plan 9 from Bell Labs**, 1995
   Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, Phil Winterbottom
   Bell Laboratories
3. **Man-Pages for Plan 9**
   `http://man.cat-v.org/plan_9`

Motivation

- UNIX, is itself an old timesharing system and has had trouble adapting to ideas born after it
- Small, cheap machines in people's offices would serve as terminals providing access to large, central, shared resources such as computing servers and file servers
- "build a UNIX out of a lot of little systems, not a system out of a lot of little UNIXes"
- Rethink UNIX abstractions, make them more general

Namespaces

- Processes have a namespace that is manipulated via `bind`, `mount` and `unmount`
- `mount` inserts a FS served by a server into the namespace
- `bind` creates an alias to an existing FS
- The server responds to requests of clients (navigate, create, remove, read, write, . . . files)
- May be local, may be on a different machine
- Every resource is a filesystem (on disk, a device, a process, env-vars, . . .)
- A filesystem consists typically of 2 files: data and ctl
- Syscalls on files provided by a server are translated into messages
- 9P is the protocol for the message exchange

The 9P protocol

```
// walks through the hierarchy to find 'wname'
// and assign it to 'newfid'
size[4] Twalk tag[2] fid[4] newfid[4]
  nwname[2] nwname*(wname[s])
size[4] Rwalk tag[2] nwqid[2] nwqid*(wqid[13])

// opens the file denoted by 'fid'
size[4] Topen tag[2] fid[4] mode[1]
size[4] Ropen tag[2] qid[13] iounit[4]

// reads 'count' bytes at 'offset' from 'fid'
size[4] Tread tag[2] fid[4] offset[8] count[4]
size[4] Rread tag[2] count[4] data[count]
```

Binding

- `bind(char *name, char *old, int flags)`
- Creates an alias of `old` as `name`
- Details depend on `flags`:
  - Replacing nodes
  - For directories: creating a union of directories (ordered)
  - What if one creates a new file in it?

    → Flag that specifies whether a dir should receive creates

    → The first one receives the file

Example

```
// replace contents at /bin with /arm/bin
bind("/arm/bin", "/bin", MREPL);
// union-mount /usr/bin *after* /bin
bind("/usr/bin", "/bin", MAFTER);
// union-mount /home/foo/bin *before* /bin
bind("/home/foo/bin", "/bin", MBEFORE);
```

Mounting

- `mount(int fd, char *path, int flags, ...)`
- Subsequent requests to `path` and below are translated into messages to `fd`

Example

```
int fd[2];
pipe(fd);

mount(fd[0], "/example", MREPL, ...);

while(1) {
  read(fd[0], ...);
  // ...
  write(fd[1], ...);
}
```

Blocking system calls

- All system calls in Plan9 are blocking
- There is no `O_NONBLOCK`
- Instead, one should use `fork` and execute the syscall in the clone
- Plan9 argues that it's both easy and efficient
- It has a special language, Alef, which makes it easy

Outline

1. Introduction
2. Multics
3. UNIX
4. Plan9
5. Conclusion

Differences in research

- None of the mentioned papers about Multics/UNIX had an evaluation
- Plan9 has a short performance evaluation and comparison with variant of UNIX
- "We will not attempt any [...] comparison with other systems, but merely note that we are generally satisfied with the overall performance of the system."

Summary

- Multics
  - Hierarchical filesystem
  - Generic I/O operations
- UNIX
  - Everything is a file
  - Simplicity
- Plan9
  - Takes the UNIX ideas even further
  - Distributed systems