Gunion: a new GEOM utility
in the FreeBSD Kernel

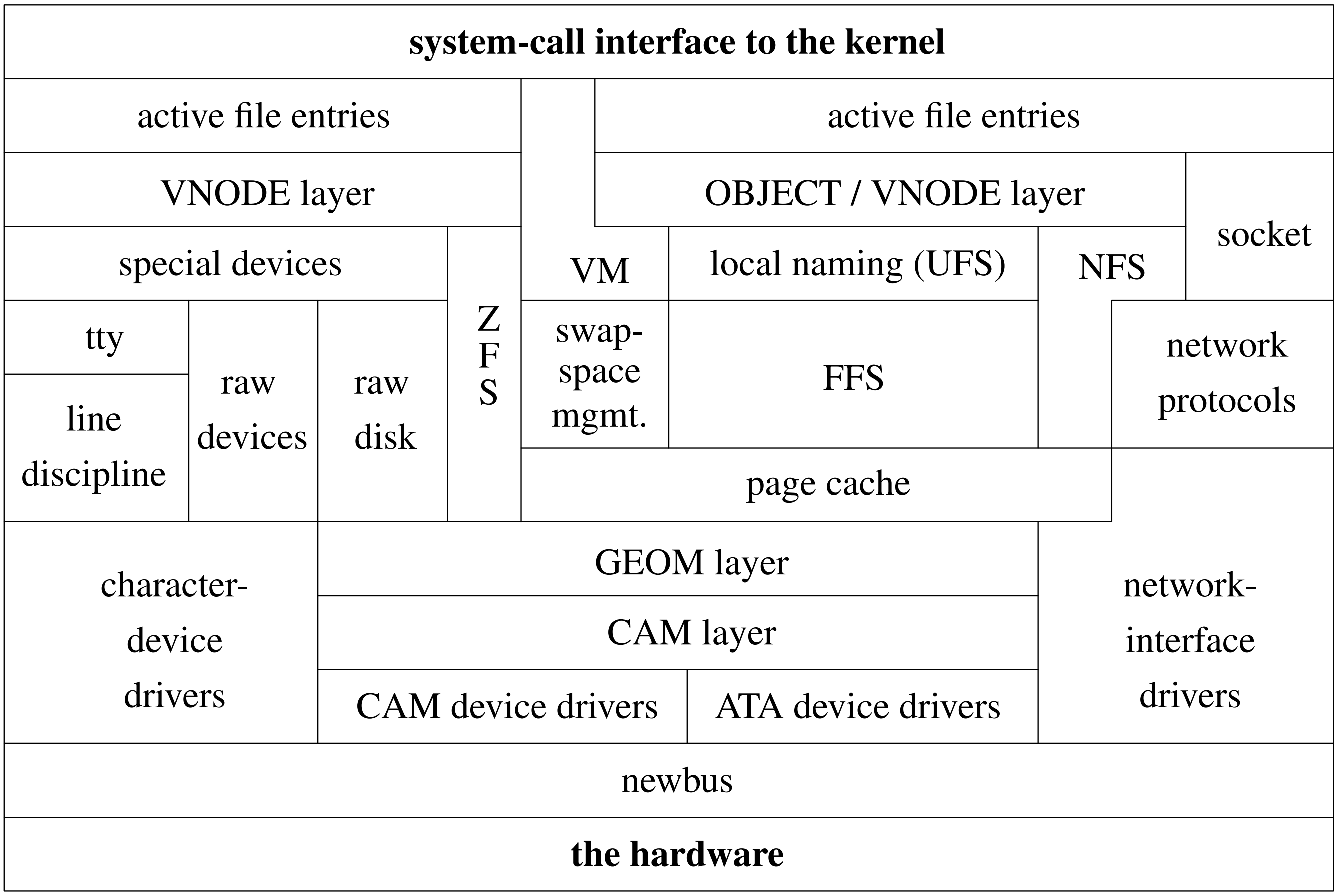
Brought to you by

Dr. Marshall Kirk McKusick

EuroBSD Conference 2023
17 September 2023

University of Coimbra
Coimbra, Portugal

Kernel I/O Structure



system-call interface to the kernel

active file entries

active file entries

VNODE layer

OBJECT / VNODE layer

socket

special devices

ZFS

VM

local naming (UFS)

NFS

tty

line discipline

raw devices

raw disk

swap-space mgmt.

FFS

network protocols

page cache

character-device drivers

GEOM layer

CAM layer

CAM device drivers

ATA device drivers

network-interface drivers

newbus

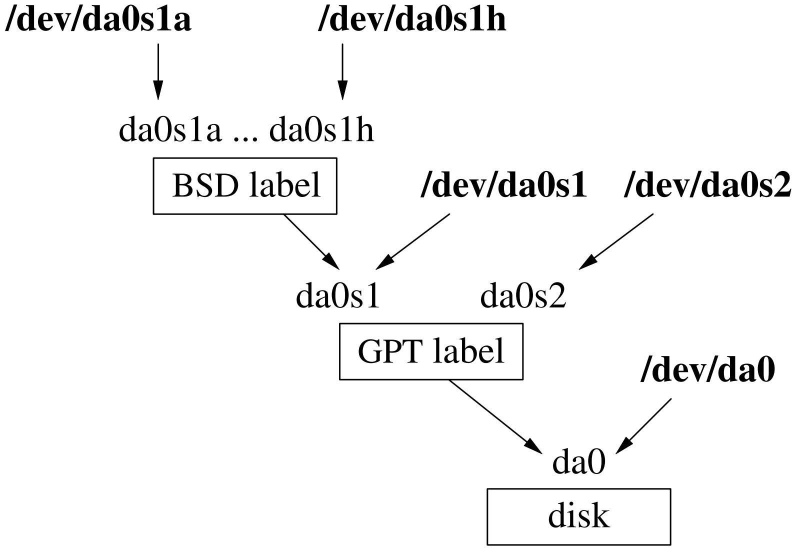
the hardware

text ref: pp. 315-316

Disk Volume Management

GEOM Layer

- disk partitioning and labelling
- disk aggregation for mirror, RAID, or striped logical volumes
- collection of I/O statistics
- I/O optimization such as disk sorting

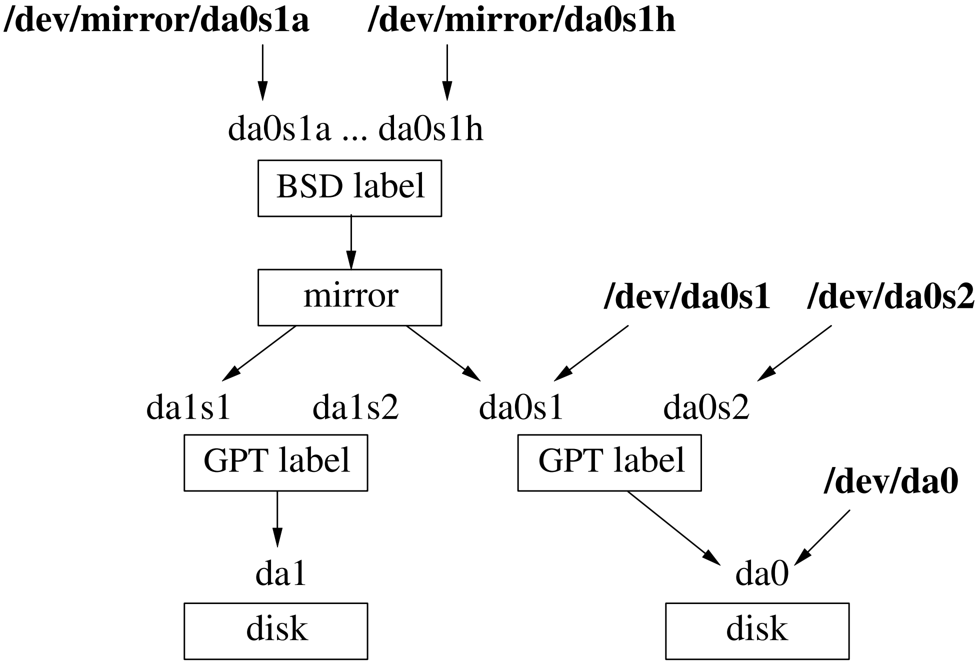


text ref: pp. 376-377, 391-399

Mirroring a Filesystem

GEOM Mirror Layer

- mirrors underlying partition
- can be inserted to copy filesystem



text ref: pp. 376-377, 391-399

GEOM Operation

- Downward requests handled by g_down
- Upward requests handled by g_up
- Modules cannot sleep or compute excessively
- Modules providing locking can allow direct dispatch
- When provider goes away, error is propagated up the stack
- When provider changes (spoiling), change is propagated up the stack

text ref: pp. 376-377, 391-399

Memory Disk

Created and controlled using mdconfig(8).

Three types of memory disks:

- Dedicated kernel memory (malloc).
- Virtual kernel memory backed by swap space (swap).
- Virtual kernel memory backed by a file (vnode).

Appears to GEOM consumers like a traditional disk.

Memory Disk Operation

Memory usage is based by type of backing.

- Malloc mode: the changes are all held in kernel memory. Size is limited to kernel memory available in a single kernel malloc().

- Swap mode: the changes are all held in the buffer cache. Pages get pushed out to the swap area when the system is under memory pressure, otherwise they stay in the kernel memory.

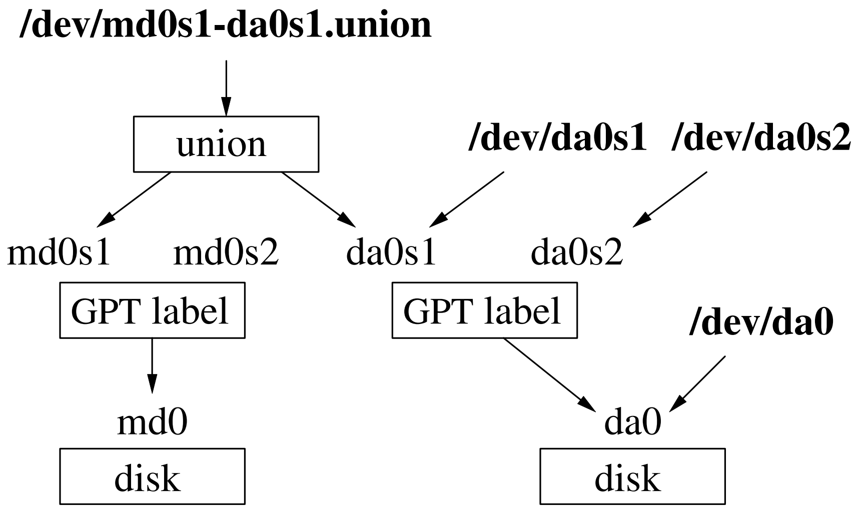
- Vnode mode: a regular file is used as backing store. Pages get pushed out to the backing file when the system is under memory pressure. All dirty pages are pushed before destroying the memory disk thus providing long-term persistence.

- For swap and vnode the space used by the memory disk is based on the amount of data written to it.

Gunion Facility

The gunion GEOM module tracks changes to a read-only disk on a writable disk.

- Logically, a writable disk is placed over a read-only disk.
- Write requests are intercepted and stored on the writable disk.
- Read requests are first checked to see if they have been written on the top (writable disk) and if found are returned.
- If they have not been written on the top disk, then they are read from the lower disk.



Gunion Commands

create – Set up a union provider on the two given devices. If the operation succeeds, the new provider appears with name /dev/(upperdev)-(lowerdev).union.

destroy – Disassemble the given union.

revert – Discard all the changes made in the top layer thus reverting to the original state of the lower device.

commit – Write all the changes made in the top device to the lower device thus committing the lower device to have the same data as the union.

Gunion Operation

The upper disk must be at least the size of the disk that it covers.

- The union metadata exists only for the period of time that the union is instantiated, so it is important to commit the updates before destroying the union.
- If the top disk has 4K sectors and is about 0.5 percent larger than the disk that it covers, it is posible (thought not currently implemented) to save the union metadata between instantiations of the union device.

Gunion Example

Create and destroy a union provider with disks /dev/da0p1 as the read-only disk on the bottom and /dev/md0 as the writable disk on the top.

```
gunion create -v md0 da0p1
mount /dev/md0-da0p1.union /mnt
```

Proceed to make changes in /mnt filesystem. If they are successful and you want to keep them.

```
umount /mnt
gunion commit -v md0-da0p1.union
```

If they are unsuccessful and you want to roll back.

```
umount /mnt
gunion revert -v md0-da0p1.union
```

When done eliminate the union.

```
umount /mnt
gunion destroy -v md0-da0p1.union
```

All uncommitted changes will be discarded when the union is destroyed.

Gunion Uses Part 1

The gunion utility can be especially useful when dealing with a large disk with a corrupted filesystem that you are unsure of how to repair.

- Use gunion to place another disk over the corrupted disk and then attempt to repair the filesystem.

- If the repair fails, ‘revert’ all the changes in the upper disk and be back to the unchanged state of the lower disk thus allowing another approach to repairing it.

- If the repair is successful ‘commit’ all the writes recorded on the top disk to be written to the lower disk.

Gunion Uses Part 2

Use the gunion utility to try out a system upgrade.

- Place the upper disk over the disk holding your filesystem that is to be upgraded.
- Run the upgrade on it.
- If it works, ‘commit’ it
- If it fails, ‘revert’ the upgrade.

GEOM NOP Module

A GEOM module that does nothing.

Was written to provide the boilerplate needed to create a GEOM module.

Features began being added to it.

- Export just a subset of the underlying provider.
- Allow forcible destroy to simulate a dying disk.
- Specify a possibly variable delay in reading and/or writing through the layer.
- Specify a probability of failure reading and/or writing through the layer.

Useful when testing:

- error recovery code.
- delay handling code.
- correctness of handing out-of-order I/O operations.

Questions

Marshall Kirk McKusick

<mckusick@mckusick.com>

http://www.mckusick.com

FreeBSD Kernel Internals on Video

This 40-hour course is the detailed version of this introductory video and provides a complete background of the FreeBSD kernel. It covers all the topics in the book. In addition, it covers other related topics including performance measurement and system tuning. The first video provides an introduction to the FreeBSD community. The remaining videos consist of fifteen lectures on the FreeBSD kernel that align with the book chapters. There are assigned readings to be completed before viewing each lecture. The first thirteen lectures have a set of exercises to be done after each video is viewed. Follow-up comments on the exercises are provided at the beginning of the lecture following the one in which they are assigned.

The syllabus for the the course is as follows:

0) Preface: an introduction to the FreeBSD community
1) Introduction: kernel terminology and basic kernel services
2) Kernel-resource management: locking
3) Processes: process structure and process management
4) Security: security framework and policies, Capsicum, and jails
5) Virtual memory: virtual-memory management, paging, and swapping
6) Kernel I/O system: multiplexing I/O, support for multiple filesystems, the block I/O system (buffer cache), and stackable filesystems
7) Devices: special files, pseudo-terminal handling, autoconfiguration strategy, structure of a disk device driver, and machine virtualization
8) Local filesystem implementation: fast filesystem (FFS)
9) Local filesystem implementation: zettabyte filesystem (ZFS)
10) Remote filesystem implementation: network filesystem (NFS)
11) Interprocess communication: concepts and terminology, basic IPC services, system layers and interfaces, and code review of a simple application that demonstrates use of the IPC and network facilities
12) Network layer: IPv4 and IPv6 protocols, firewalls, and routing
13) Transport layer: TCP and SCTP
14) System startup: boot loaders, kernel startup, and system launch; system measurement tools
15) System tuning: performance measurement and system tuning

In addition to the preface and fifteen lecture videos, you also receive a copy of the course notes containing copies of all the overhead slides used in the course, an extensive set of written notes about each lecture, a set of weekly readings from this textbook, thirteen sets of exercises (along with answers), and a set of papers that provide supplemental reading to the text.

Tiered pricing is available for companies, individuals, and students. On-site courses can be arranged. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

Advanced FreeBSD Course on Video

The 46-hour course provides an in-depth study of the source code of the FreeBSD kernel. It is aimed at users who already have a good understanding of the algorithms used in the FreeBSD kernel and want to learn the details of each algorithm's implementation. Students are expected to have either taken this class or a similar class taught by the instructor or to have throughly read and understood "The Design and Implementation of the FreeBSD Operating System, Second Edition" (published by Pearson Education's Addison-Wesley Professional division). They are also expected to have a complete background in reading and programming in the C programming language. Students will not need to prove relationship with a source license holder, as the course is based on the non-proprietary kernel sources released by the FreeBSD project.

The class consists of fifteen lectures on the FreeBSD kernel source code. The lecture topics are:

1) Organization, overview of source layout
2) Kernel header files
3) System calls and file opening
4) Pathname translation and file creation
5) Vnode interface mechanics, writing to an FFS file
6) Write to a ZFS file
7) Opening, using, and closing locally connected sockets
8) User datagram protocol and routing
9) TCP algorithms
10) Fork, exit, and exec
11) Signal generation and delivery, scheduling
12) Virtual memory header files, file mapping
13) Page fault service, pageout processing
14) NFS client and server operation
15) Multiplexing with select, system startup

In addition to the fifteen lecture videos, you also receive a CD-ROM with a copy of the FreeBSD kernel source covered in the lectures and a copy of the lecture notes.

Tiered pricing is available for companies, individuals, and students. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

FreeBSD Networking from the Bottom Up on Video

This course describes the FreeBSD networking stack. It is made up of a series of lectures derived from tutorials given by George Neville-Neil.

The class currently consists of five lectures, though additional lectures are being developed. The current lecture topics are:

1) Device Drivers: how to write and maintain network drivers in FreeBSD. By way of example it uses the Intel Gigabit Ethernet driver (*igb*). The lecture covers the basic data structures and APIs necessary to implement a network driver in FreeBSD. It is specific enough that given a device and a manual, you should be able to develop a working driver on your own.
2) The IPv6 Stack: an in-depth discussion and code walk-through of version 6 of the IP protocols, describing and dissecting the paths that packets take from the driver layer up to the socket layer of the network stack. The lecture covers the four paths packets travel through the network stack: reception, transmission, forwarding, and error handling.
3) Routing: packet forwarding and routing subsystems in FreeBSD. The routing and forwarding code are the glue that keeps the networking stack together, connecting the network protocols, such as IPv4 and IPv6, to their underlying data link layers and making sure that packets are sent to the correct next hop in the network. Topics in the lecture include the Routing Information Base (RIB), Forwarding Information Base (FIB), and the systems that interact with them. Also covered are routing sockets and the RIB/FIB APIs, the address-resolution protocol (ARP), Neighbor Discovery (ND6), the Common Address Redundancy Protocol (CARP), the IP firewall and traffic shaper control program (*ipfw*), and the packet filter interface (*pfil*).
4) Packet Processing Frameworks: The FreeBSD Kernel has several different packet processing frameworks—software that is meant to transform packets but which are not traditionally considered to be network protocols. It is these packet processing frameworks that are often the basis for new products built with FreeBSD. This lecture covers all of the packet processing frameworks, including the Berkeley Packet Filter (BPF), IP Firewall (IPFW), Dummynet, Packet Filter (PF), Netgraph, and netmap. It discusses the appropriate use of each framework and takes a walk through the relevant sections of each framework. Working examples of extensions to each framework are given so that students can see how to build new systems with and around the frameworks that are present in the kernel.
5) A Look Inside FreeBSD Using DTrace. DTrace is a modern system that gives software developers the ability to add low overhead tracing that is always available to programs that they are creating, modifying, and debugging. The desired tracing is described and controlled with an advanced scripting language. This tutorial covers the basics of DTrace, including basic and advanced uses. Using a set of worked examples, students learn to add tracing to user space and kernel space systems. The tutorial includes a set of short labs carried out on virtual machines that give the students hands-on experience working with DTrace.

Each lecture may be purchased separately and comes with a copy of its course notes. Tiered pricing is available for companies, individuals, and students. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

CSRG Archive CD-ROMs

Thanks to the efforts of the volunteers of the “UNIX Heritage Society” (see http://www.tuhs.org) and the willingness of Caldera to release 32/V under an open source license (see http://www.mckusick.com/csrg/calder-lic.pdf), it is now possible to make the full source archives of the University of California at Berkeley’s Computer Systems Research Group (CSRG) available.

The archive contains four CD-ROMs with the following content:

CD-ROM #1—Berkeley Systems 1978–1986

1bsd
2.10
2.79
2.8
2.9
2.9pucc
2bsd
3bsd
4.0
4.1
4.1.snap
4.1a
4.1c.1
4.1c.2
4.2
4.2buglist
4.3
VM.snapshot.1
pascal.2.0
pascal.2.10

CD-ROM #2—Berkeley Systems 1987–1993

4.3reno
4.3tahoe
4.4BSD-Lite1
VM.snapshot.2
net.1
net.2

CD-ROM #3—Final Berkeley Releases

4.4
4.4BSD-Lite2

CD-ROM #4—Final /usr/src including SCCS files

Contrib
Makefile
README
SCCS
admin
bin
contrib
etc
games
include
lib
libexec
local
old
sbin
share
sys
usr.bin
usr.sbin

The University of California at Berkeley wants you to know that these CD-ROMs contain software developed by the University of California at Berkeley and its many contributors.

The CD-ROMs are produced using standard pressing technology, *not* with write-once CD-R technology. Thus, they are expected to have a 100-year lifetime rather than the 10–20 years expected of CD-R disks. The CDs are sold only in complete sets; they are not available individually. The price for the 4-CD set is $99. The contents of the original four CD-ROMs plus some additional early UNIX distributions is available on a single DVD using 100-year lifetime M-DISC technology for $149.00. The archive can be ordered from

http://www.mckusick.com/csrg/

The compilation of this archive is copyright © 1998 by Marshall Kirk McKusick. You may freely redistribute it to anyone else. However, I appreciate you buying your own copy to help cover the costs that I incurred in producing the archive.

History of UNIX at Berkeley

Learn the history of the BSD (Berkeley Software Distributions) from one of the key developers who brings the history to life, complete with anecdotes and interesting footnotes to the historical narrative.

Part I is titled “Twenty Years of Berkeley UNIX: From AT&T-Owned to Freely Redistributable.” The history of UNIX development at Berkeley has been recounted in detail by Marshall Kirk McKusick in his chapter in the O’Reilly book Open Sources: *Voices from the Open Source Revolution* and is now recounted in part one of this video. It begins with the start of the BSD community at the University of California at Berkeley in the late 1970s. It relates the triumphs and defeats of the project and its releases during its heydays in the 1980s. It concludes with the tumultuous lawsuit ultimately settled in Berkeley’s favor, which allowed the final release in 1992 of 4.4BSD-Lite, an open-source version of BSD.

Part II is titled “Building and Running An Open-Source Community: The FreeBSD Project.” It tells the story of the independent development by the FreeBSD project starting from the open-source release from Berkeley. The FreeBSD project patterned its initial community structure on the development structure built up at Berkeley. It evolved and expanded that structure to create a self-organizing project that supports an ever growing and changing group of developers around the world. This part concludes with a description of the roles played by the thousands of volunteer developers that make up the FreeBSD Project of today.

Dr. Marshall Kirk McKusick’s work with UNIX and BSD development spans over thirty years. It begins with his first paper on the implementation of Berkeley Pascal in 1979, goes on to his pioneering work in the eighties on the BSD Fast File System, the BSD virtual memory system, and the final release of 4.4BSD-Lite from the University of California Berkeley Computer Systems Research Group. Since 1993, he has been working on FreeBSD, adding soft updates, snapshots, and the second-generation Fast Filesystem to the system. A key figure in UNIX and BSD development, his experiences chronicle not only the innovative technical achievements, but also the interesting personalities and philosophical debates in UNIX since its inception in 1970.

The price for the video is $19.95. The video can be ordered from

http://www.mckusick.com/history/