Getting Extended Error Messages from the FreeBSD Kernel

Brought to you by

Dr. Marshall Kirk McKusick

BSD Canada Conference 2026
June 20, 2026

University of Ottawa
Ottawa, Canada

Introduction

- Reporting on work done by Konstantin Belousov (kib@freebsd.org) under sponsorship of the FreeBSD Foundation.

- My kernel extended-error message work has been sponsored by Netflix.

- Historically kernel just returned a defined set of error numbers.

- There are often multiple reasons for a given error number. For example, the mmap(2) manual page lists sixteen reasons for returning EINVAL.

- Want a way to return more complete error information.

- Extended errors allow more specific error information to be returned.

Historical Reason for Error Number Return

- UNIX was developed on 16-bit PDP-11s
  - Only room for 64K of text and 64K of data
  - About 10K lines of code
  - Strings use scarce space
  - Much more compact to just return a number
  - Number can be expanded to a string in user space

Added Size of Kernel Strings

- FreeBSD is used in embedded systems
  - Often have limited resouces (CPU speed, memory, storage)
  - Run dedicated application without user display
  - No need for extended error messages
  - Save space by excluding extended error strings
- Original option in kernel configuration file
  - BLOAT_KERNEL_WITH_EXTERR
  - changed to EXTERR_STRINGS
- Effect of eliminating external error strings
  - Currently 2K bytes
  - With additional use may grow considerably
  - Example, KASSERT adds 676K of strings

Need for Kernel Extended Error Messages

- Some error codes are heavily used
  - When the error is returned it can be difficult to understand what caused it.
  - An example is the EINVAL (Invalid argument) error that is returned from nearly 14000 places in the amd64 kernel.
  - It is returned for sixteen different reasons in the mmap() system call.
- Currently only slightly over 200 extended error messages in the kernel. Many more are needed and will hopefully be added.

Kernel Setup of Extended Error Messages

- Each kernel file that wants to produce extended errors must register itself in `<sys/exterr_cat.h>` then defines its `EXTERR_CATEGORY`.
  - The mmap() system call is implemented in vm/vm_mmap.c
  - First step is to add
    `#define EXTERR_CAT_MMAP 1`
    in
    `<sys/exterr_cat.h>`
    where the number is the next available value.
  - In the includes at the top of sys/vm/vm_mmap.c add

```
#define EXTERR_CATEGORY EXTERR_CAT_MMAP
#include <sys/exterrvar.h>
```

    Note that order is important.

Kernel Use of Extended Error Messages

- In places where it formerly returned an error number it uses EXTERROR instead
  - Formerly

```
if ((prot & ~(_PROT_ALL |
     PROT_MAX(_PROT_ALL))) != 0) {
         return (EINVAL);
}
```

  - Becomes

```
if ((prot & ~(_PROT_ALL |
    PROT_MAX(_PROT_ALL))) != 0) {
         return (EXTERROR(EINVAL,
             "unknown PROT bits %#jx", prot));
}
```

  - Each error may have up to two values for its message.

Requesting Extended Error Messages

- Each thread in a process must register a structure to use for its extended error messages.
- By default the C runtime sets up a structure when creating new threads:
  - The C library sets the buffer for the initial thread.
  - The threading library (libthr) sets the buffer for a new thread as part of its creation.
  - Language runtimes that do not use the C runtime such as “go” do not currently use the facility.

Returning Extended Error Messages

- If a system call generates an extended error message, its details are copied into the registered structure.
- No overhead for successful system calls.
- No overhead for threads not wanting extended error messages (indicated by deleting their extended error structure).
- Extended error messages are constructed from the user supplied structure and printed by the err(3) routines.
- Extended error messages are also shown by kdump(1) when displaying output from a ktrace(1) log.

Displaying Extended Error Messages

```
/* open file to be mapped */
filename = av[1];
if ((fd = open(filename,
    O_CREAT|O_RDWR, 0664)) < 0)
        err(2, "open %s failed", filename);
/* find size of file to be mapped */
if ((fstat(fd, &st)) < 0)
        err(4, "stat %s failed", filename);
len = st.st_size;
prot = PROT_READ | PROT_WRITE;
mapping = MAP_FILE | MAP_NOSYNC;
if ((mapaddr = mmap(0, len, mapping, prot,
    fd, 0)) == -1)
        err(1, "mmap %s failed", filename);
```

chez % ./examples bigfile
mmap bigfile failed: Invalid argument
(unknown PROT bits 0x800)
chez % setenv EXTERROR_VERBOSE brief
chez % ./examples bigfile
mmap bigfile failed: Invalid argument
(unknown PROT bits 0x800
(src sys/vm/vm_mmap.c:200))

Displaying Extended Error Messages (cont)

```
offset = 0;
if ((mapaddr = mmap(0, len, PROT_READ,
    MAP_ANON, fd, offset)) == -1)
        err(1, "mmap %s failed", filename);
```

chez % ./examples bigfile
mmap bigfile failed: Invalid argument
(fd 3 not -1 for MAP_ANON)

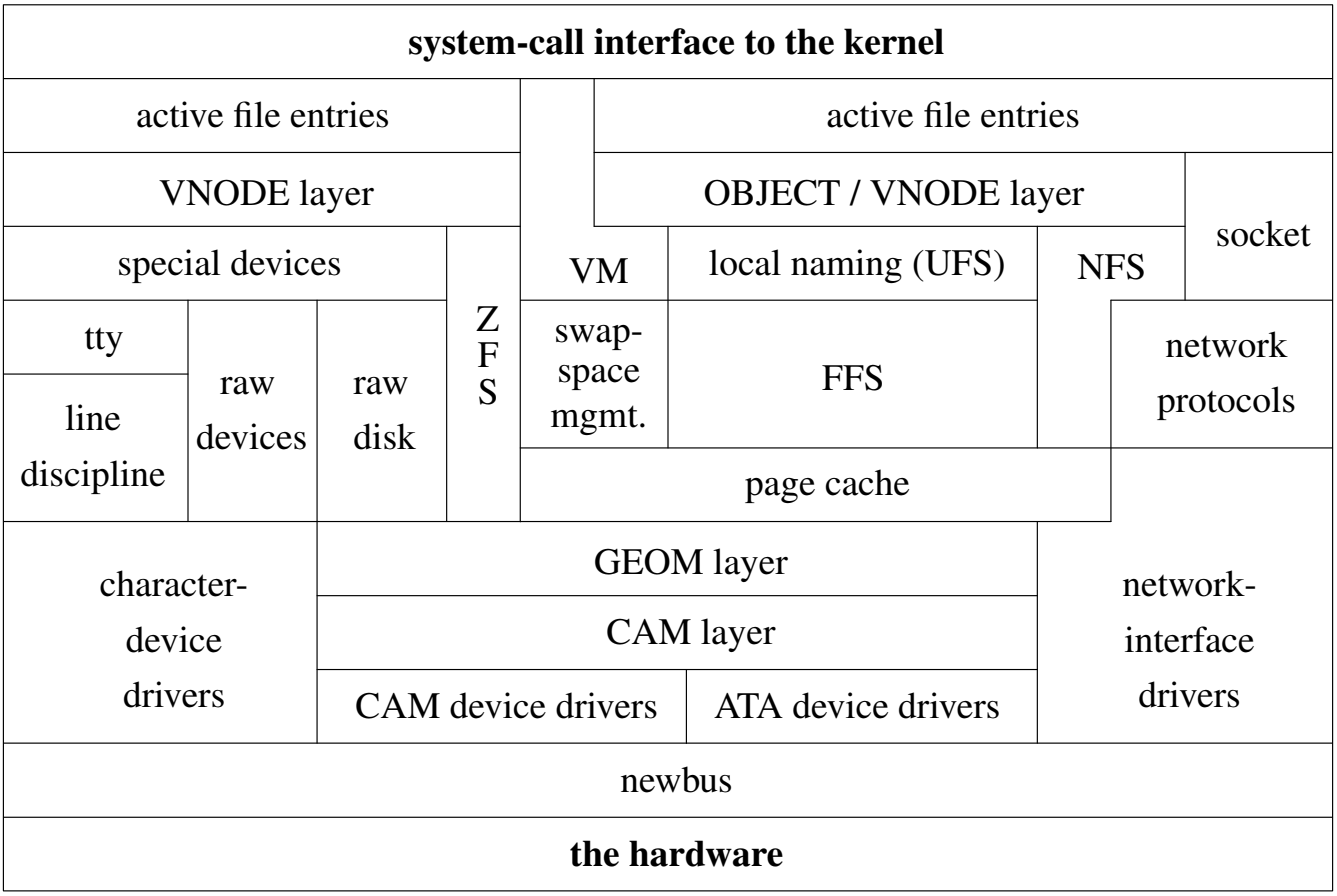
```
if ((mapaddr = mmap(0, len, PROT_READ,
    MAP_STACK, fd, offset)) == -1)
        err(1, "mmap %s failed", av[1]);
```

chez % ./examples bigfile
mmap bigfile failed: Invalid argument
(MAP_STACK with prot 0x1 < rw)

```
if ((mapaddr = mmap(0, len, PROT_READ,
    MAP_SHARED | MAP_PRIVATE,
    fd, off set)) == -1)
        err(1, mmap %s failed", av[1]);
```

chez % ./examples bigfile
mmap bigfile failed: Invalid argument (both
SHARED and PRIVATE set (flags 0x3))

Kernel I/O Structure



- GEOM Operation
  - Downward requests handled by g_down
  - Upward requests handled by g_up
  - Modules providing locking can allow direct dispatch

How I/O Requests Are Made

- Allocate a buffer from which the request will be done
  - Empty buffer for a read
  - Filled buffer for a write
- Create a bio structure to describe the I/O
  - Contains a pointer to the buffer data and the transfer size
  - Contains a pointer to geom entity (typically a disk) on which to do the I/O and the block at which the I/O begins.
- Request may be handled by direct dispatch (same thread) or by the g_down thread.
- Response may be handled by disk interrupt thread or by the g_up thread, or by the requesting thread.

Extended Errors from Async Kernel Threads

- Extended errors require a user-process in which they can be reported.
  - Only user-process threads doing system calls have a context in which extended errors can be reported.
  - Kernel worker threads never return to a user context, so need a way to pass extended errors to the user-thread for which they have been working.
- Provide a mechanism to let extended errors get passed between kernel threads
  - Add an extended error structure in the buffer and bio structures to record any extended errors.
  - When I/O completes with an error, the contents of the bio or buf extended error structure is copied to the calling threads extended error structure.
  - Allows the widely generated EIO error to explain the cause of the error.

Extended Error Status

- Framework is fully in place.
- Folks that have started adding extended errors

  Konstantin Belousov (kib@freebsd.org)
  vm and other subsystems

  Andre Silva <andasilv@amd.com>
  EINVAL returns in ibs_allocate_pmc()

  Ricardo Branco <rbranco@suse.de>
  fcntl's

  Mark Johnston <markj@FreeBSD.org>
  several kernel subsystems

  Gleb Popov <arrowd@FreeBSD.org>
  fuse code

  Lexi Winter <ivy@FreeBSD.org>
  network bridge code

  Warner Losh <imp@FreeBSD.org>
  various I/O subsystems

Questions

Marshall Kirk McKusick

email: <mckusick@mckusick.com>

web site: www.mckusick.com

YouTube:
youtube.com/@marshallkirkmckusick1756

Thanks to Netflix for their support for me on this project.