BSD Driver Harmony

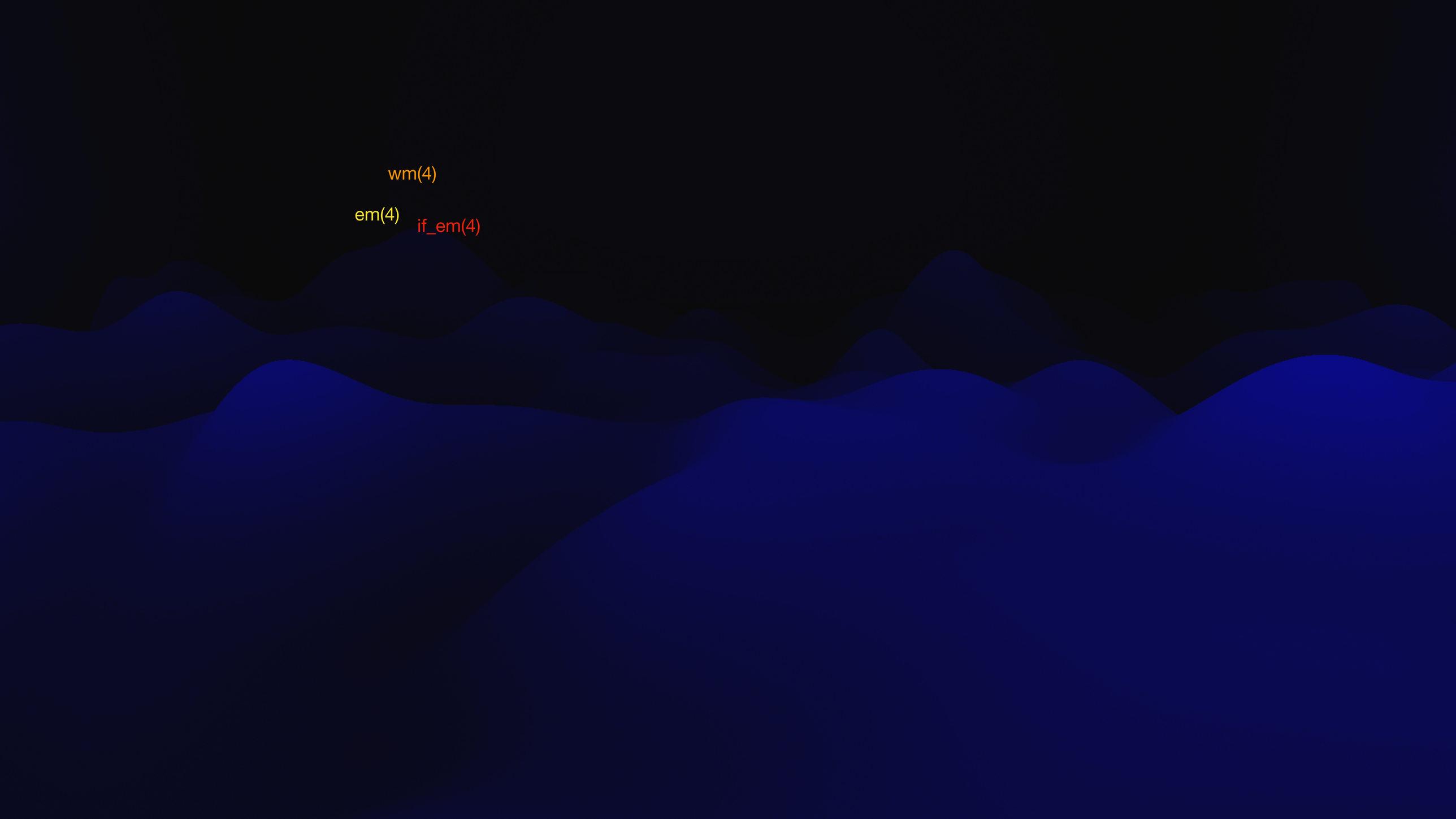
BSD devroom - FOSDEM 2023

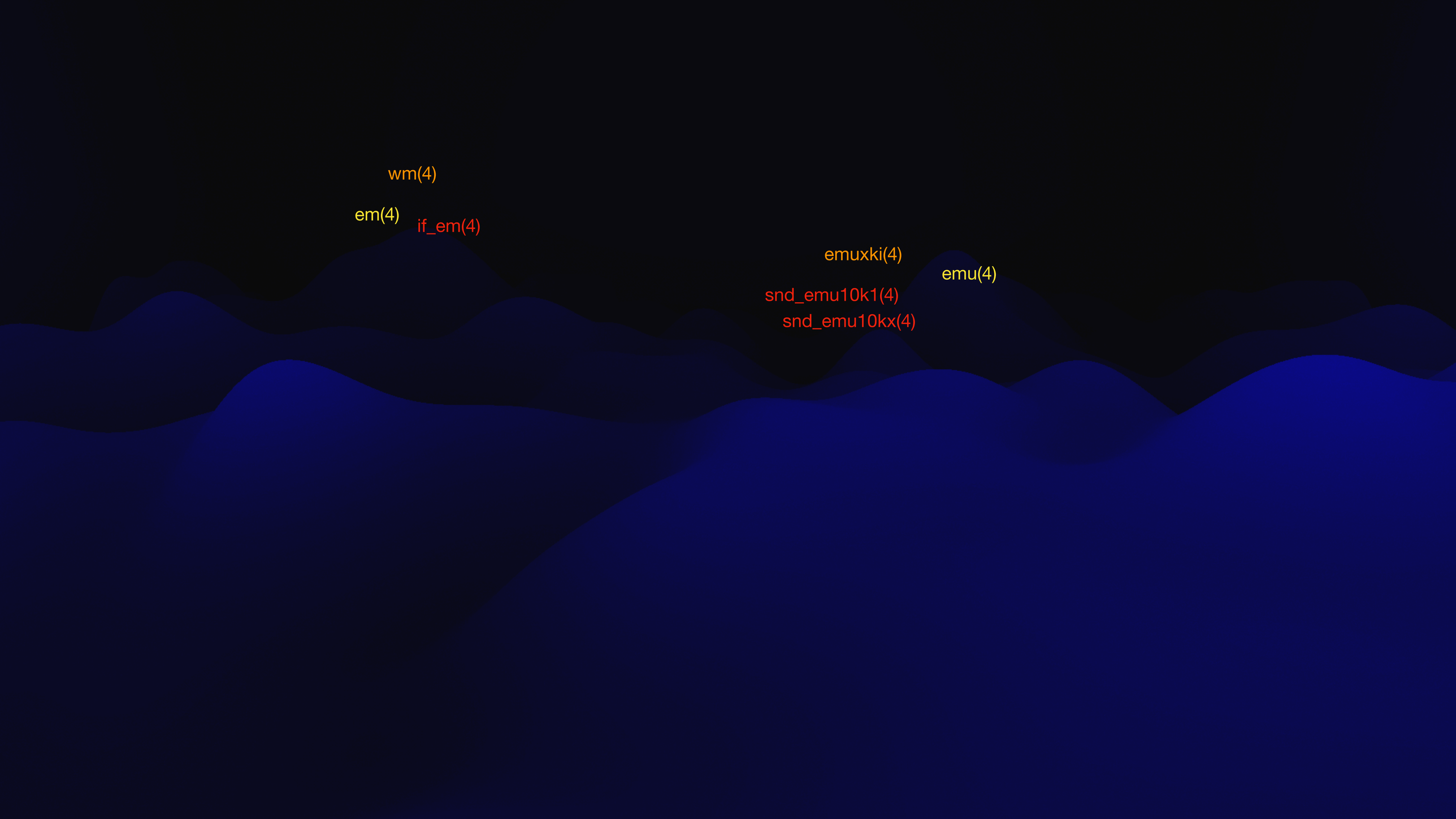
Pierre Pronchery <khorben@NetBSD.org> - February 4th 2023

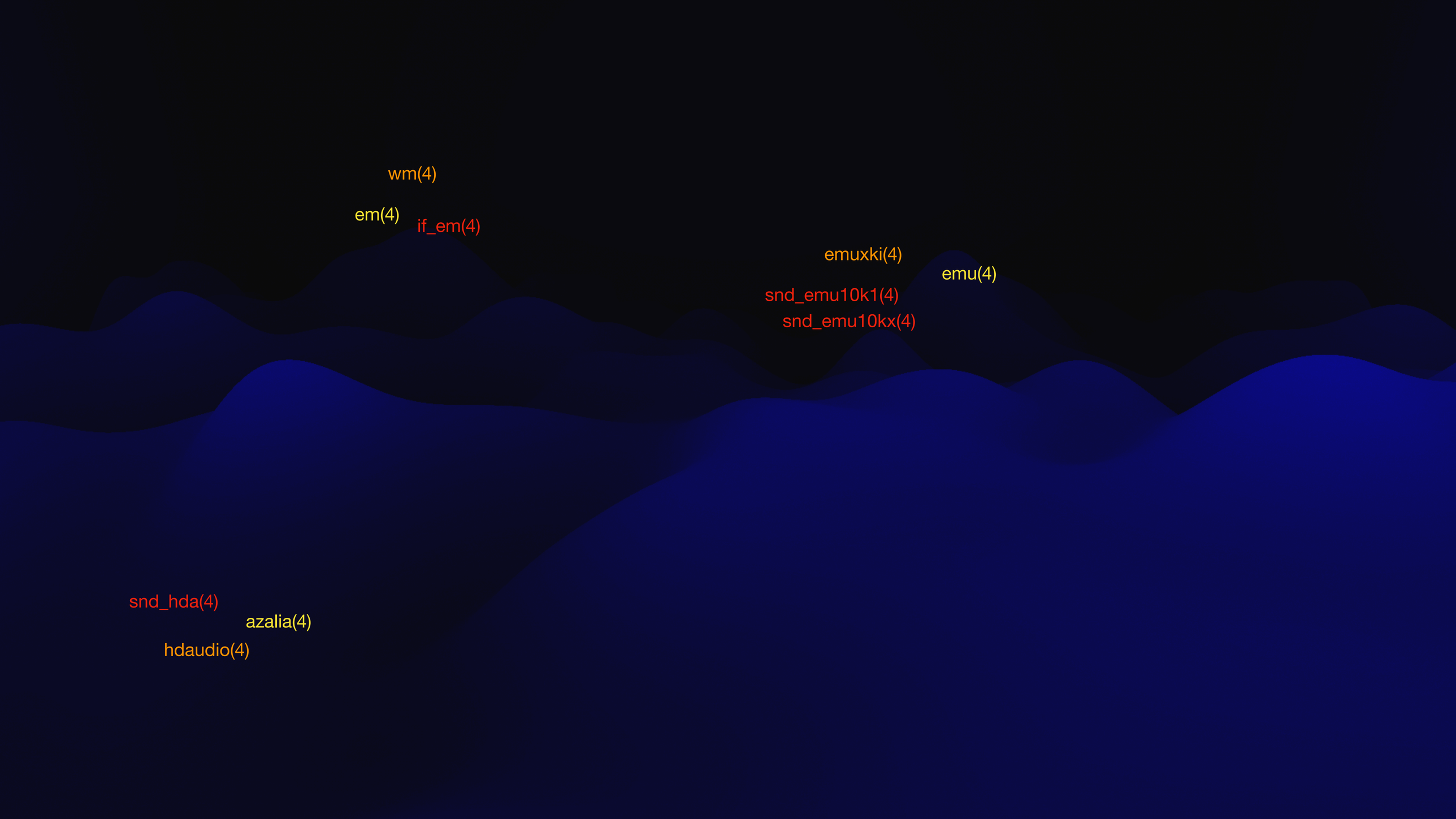
BSD Driver Harmony

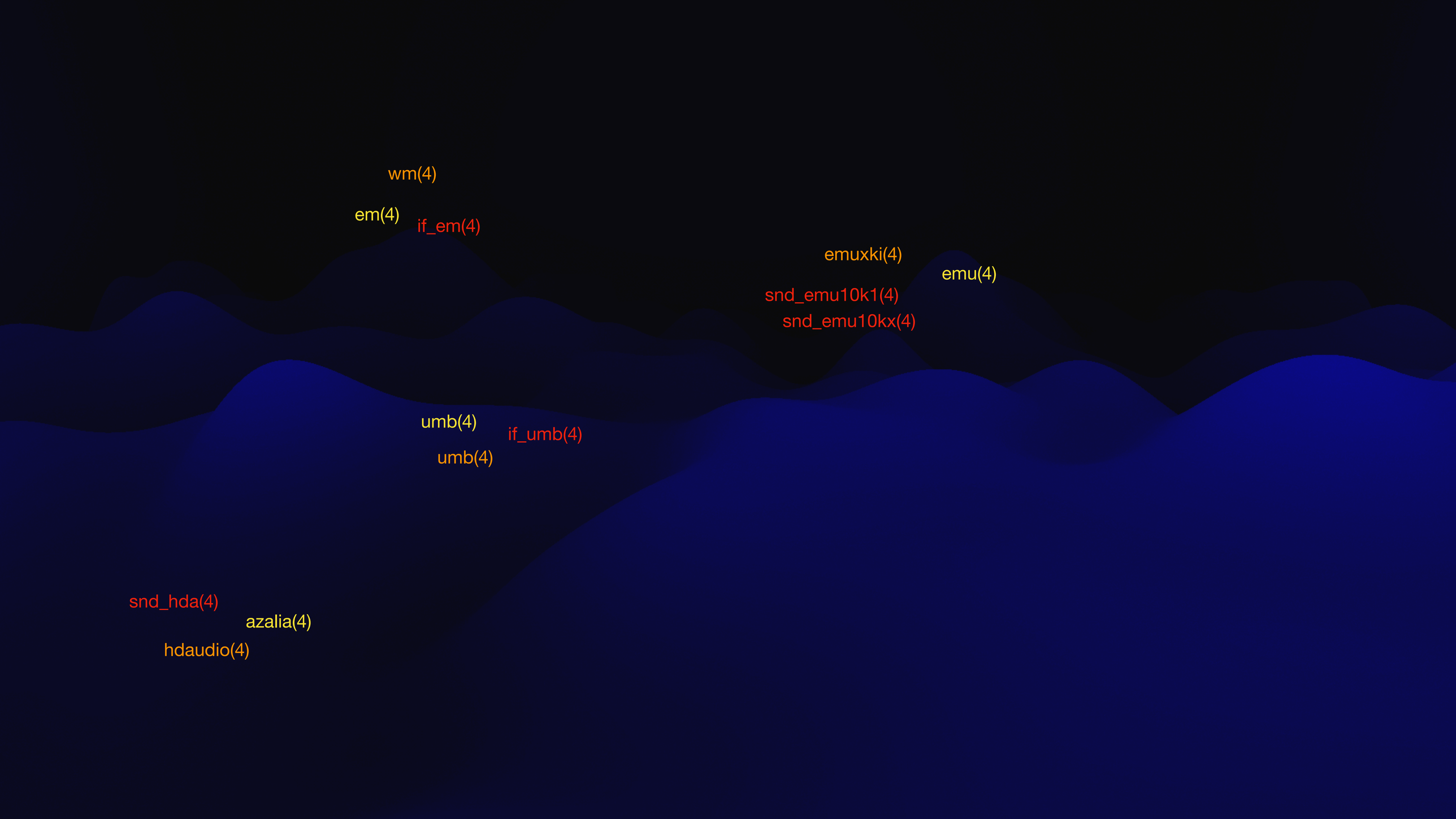
Agenda

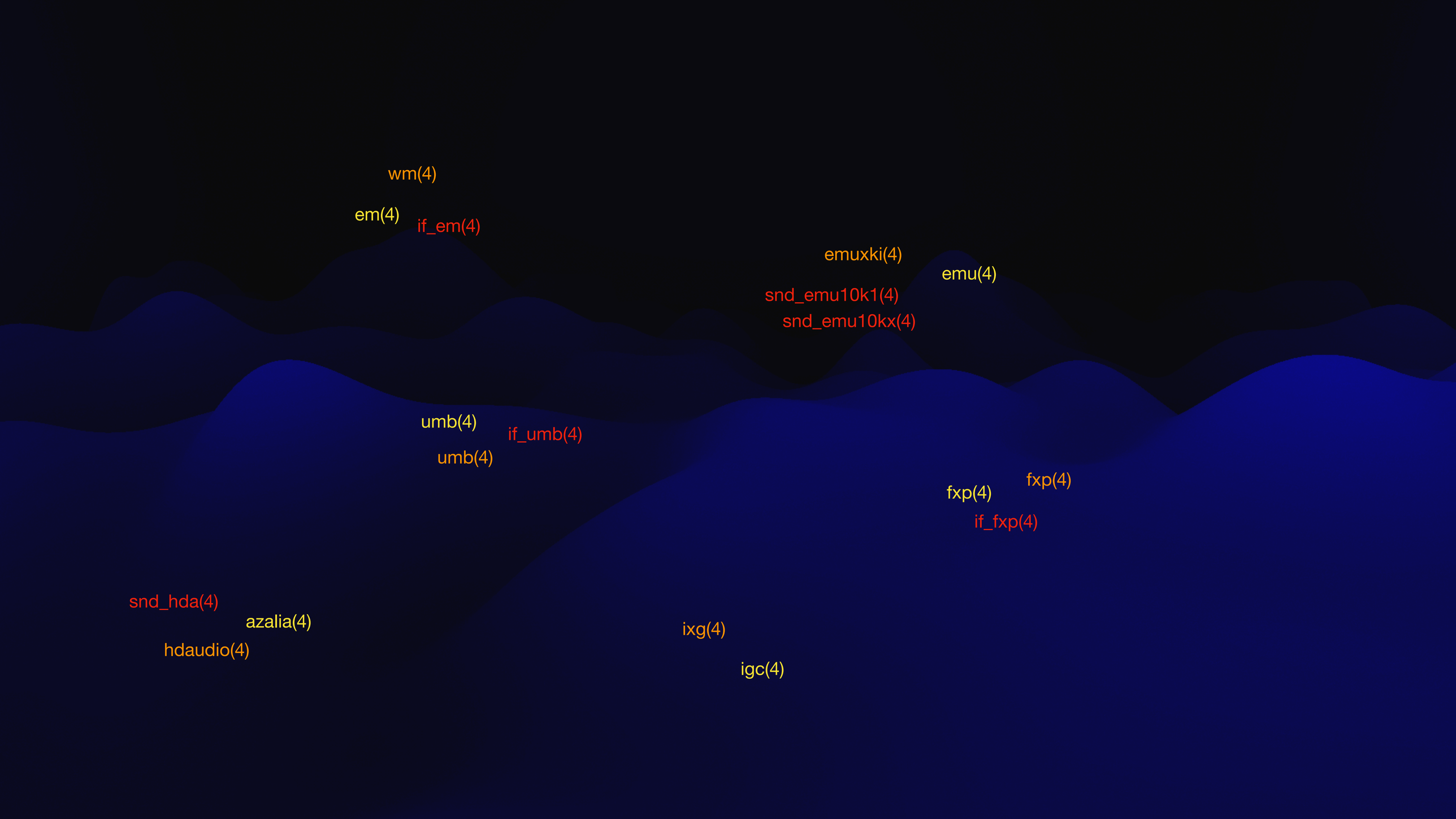
1. Background
2. Comparing device driver code
3. Call for collaboration
    1. Driver code
    2. Beyond
    3. Commitee

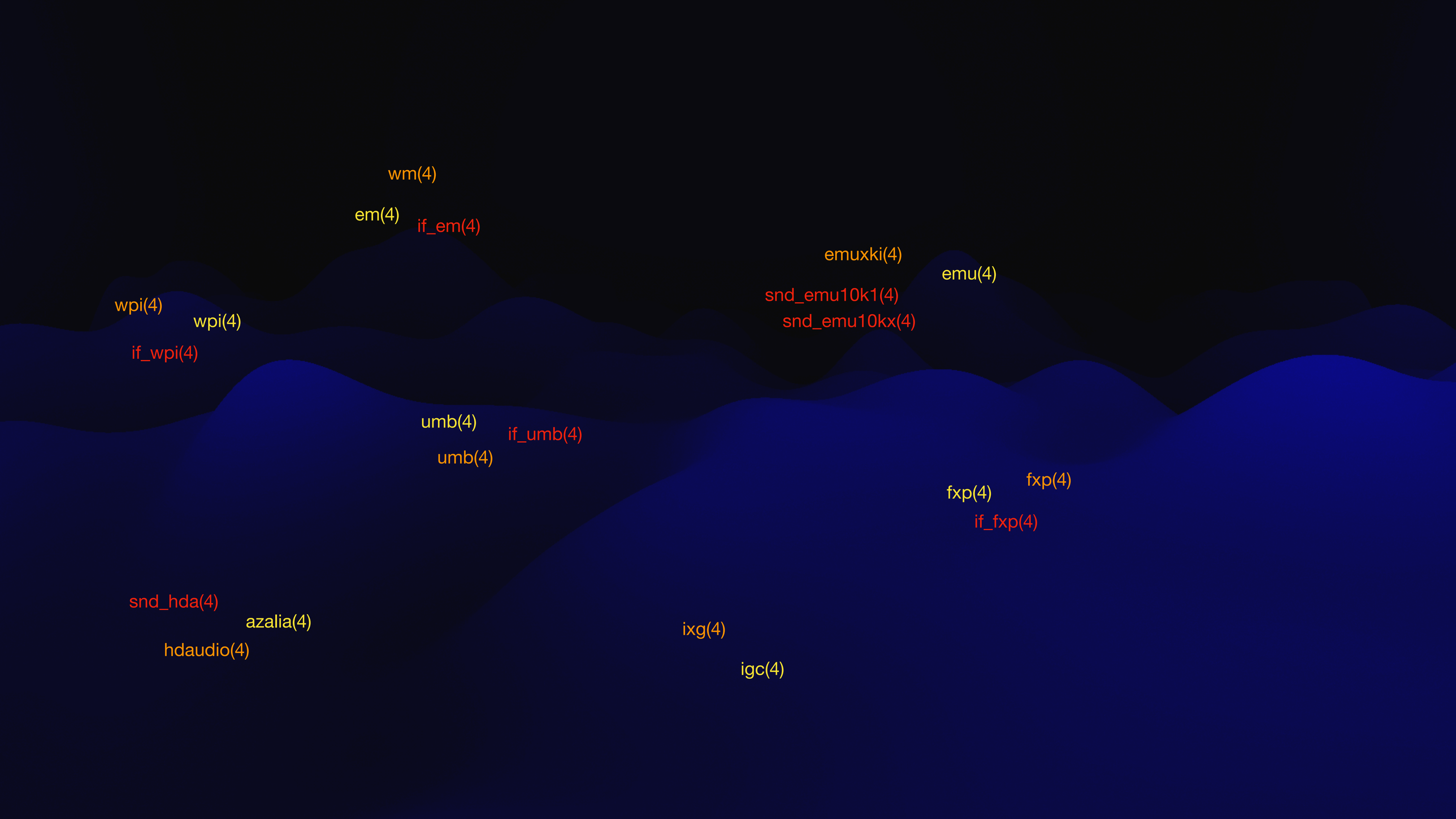

wm(4)
em(4)
if_em(4)

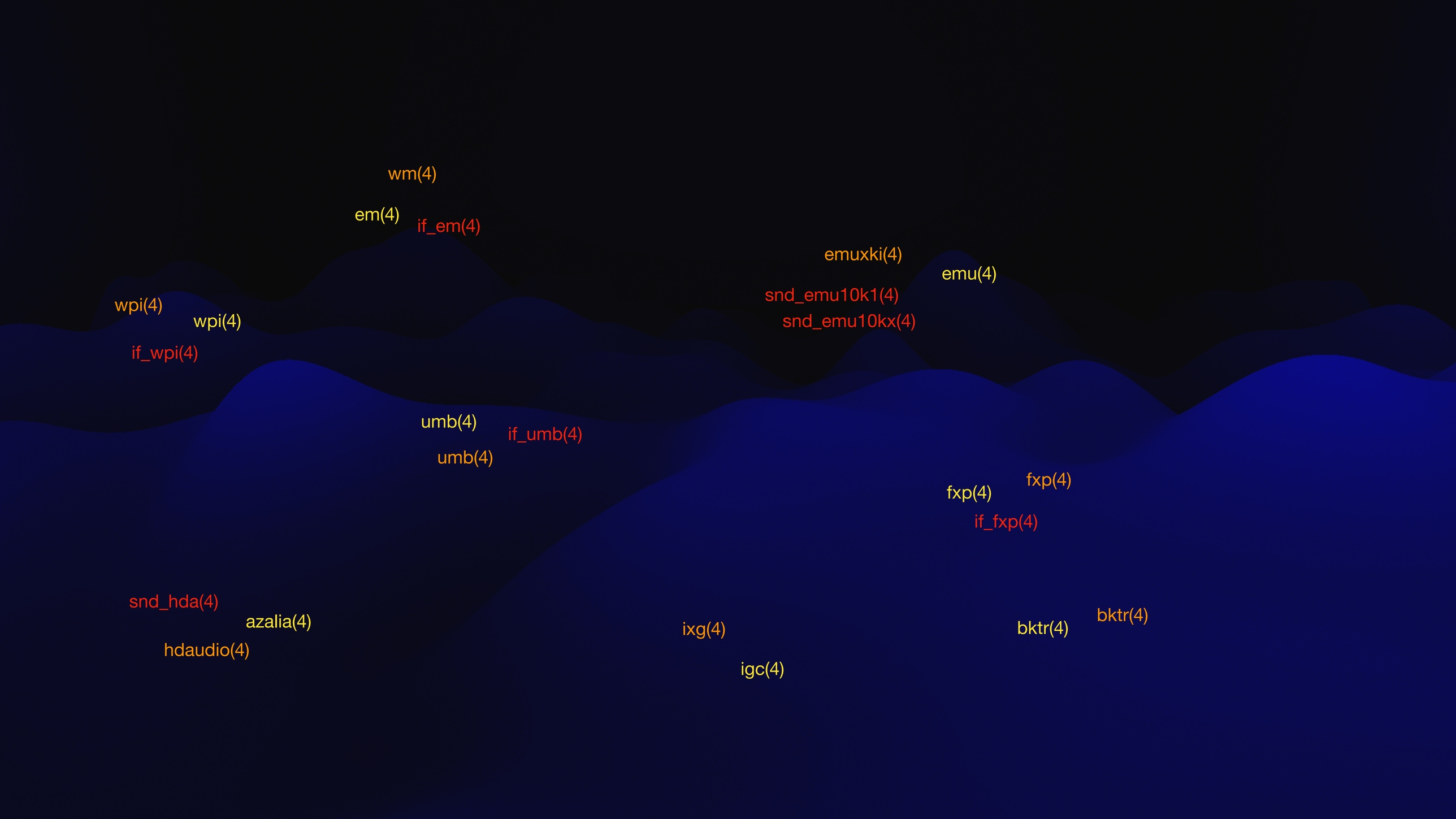

wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)

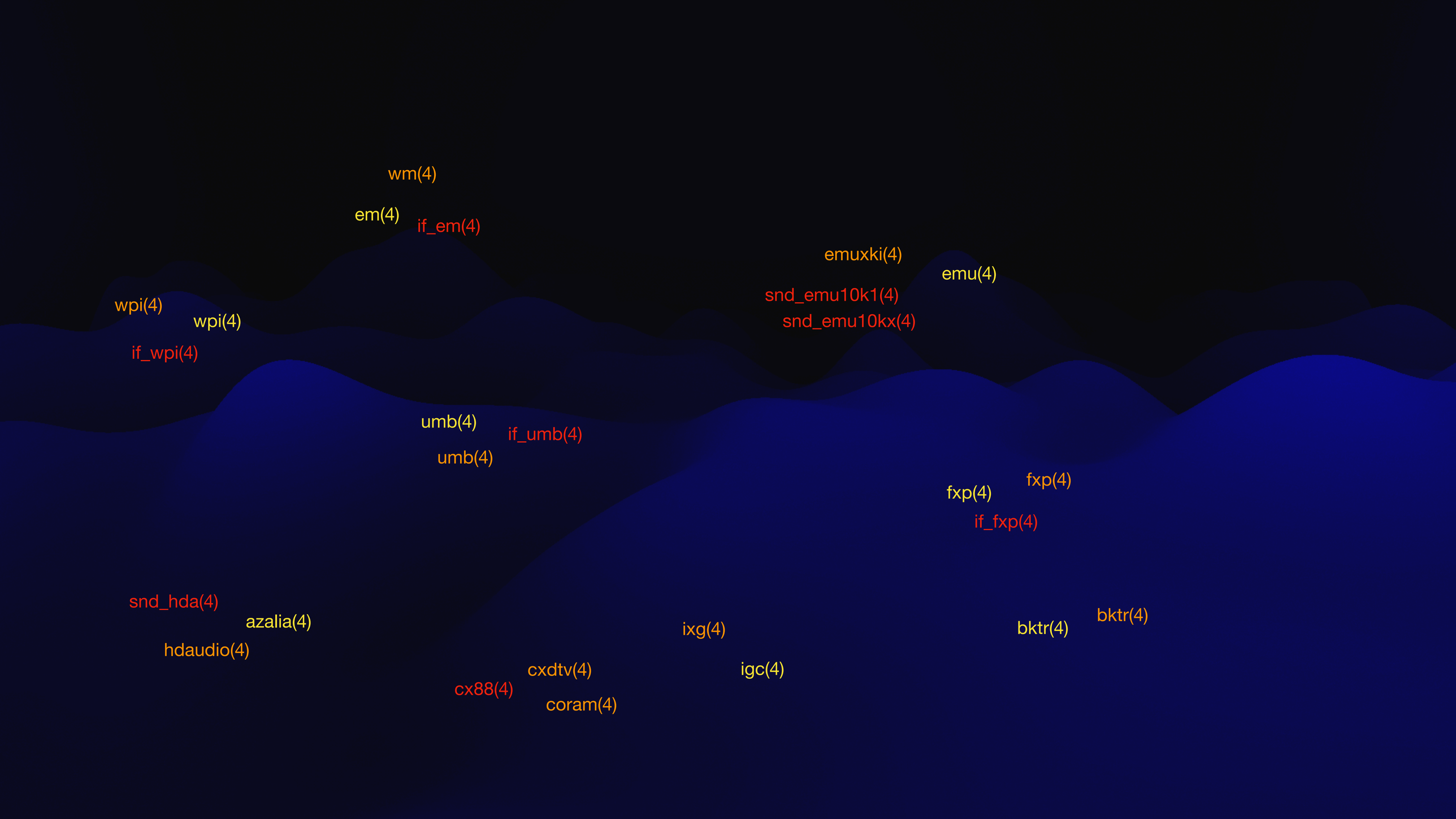

wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
snd_hda(4)
azalia(4)
hdaudio(4)


wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
umb(4)
if_umb(4)
umb(4)
snd_hda(4)
azalia(4)
hdaudio(4)


wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
umb(4)
if_umb(4)
umb(4)
fxp(4)
fxp(4)
if_fxp(4)
snd_hda(4)
azalia(4)
hdaudio(4)
ixg(4)
igc(4)


wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
wpi(4)
wpi(4)
if_wpi(4)
umb(4)
if_umb(4)
umb(4)
fxp(4)
fxp(4)
if_fxp(4)
snd_hda(4)
azalia(4)
hdaudio(4)
ixg(4)
igc(4)


wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
wpi(4)
wpi(4)
if_wpi(4)
umb(4)
if_umb(4)
umb(4)
fxp(4)
fxp(4)
if_fxp(4)
snd_hda(4)
azalia(4)
hdaudio(4)
ixg(4)
igc(4)
bktr(4)
bktr(4)


wm(4)
em(4)
if_em(4)
emuxki(4)
emu(4)
snd_emu10k1(4)
snd_emu10kx(4)
wpi(4)
wpi(4)
if_wpi(4)
umb(4)
if_umb(4)
umb(4)
fxp(4)
fxp(4)
if_fxp(4)
snd_hda(4)
azalia(4)
hdaudio(4)
ixg(4)
igc(4)
bktr(4)
bktr(4)
cxdtv(4)
cx88(4)
coram(4)

Background

Kind of a mess?

BSD Drivers

Rabbit hole?

NAME

driver – structure describing a device driver

SYNOPSIS

```
#include <sys/param.h>
#include <sys/kernel.h>
#include <sys/bus.h>
#include <sys/module.h>

static int foo_probe(device_t);
static int foo_attach(device_t);
static int foo_detach(device_t);
static int foo_frob(device_t, int, int);
static int foo_twiddle(device_t, char *);

static device_method_t foo_methods[] = {
        /* Methods from the device interface */
        DEVMETHOD(device_probe,         foo_probe),
        DEVMETHOD(device_attach,        foo_attach),
        DEVMETHOD(device_detach,        foo_detach),

        /* Methods from the bogo interface */
        DEVMETHOD(bogo_frob,            foo_frob),
        DEVMETHOD(bogo_twiddle,         foo_twiddle),

        /* Terminate method list */
        DEVMETHOD_END
};

static driver_t foo_driver = {
        "foo",
        foo_methods,
        sizeof(struct foo_softc)
};

static devclass_t foo_devclass;

DRIVER_MODULE(foo, bogo, foo_driver, foo_devclass, NULL, NULL);
```

```
DRIVER(9)                  Kernel Developer's Manual                  DRIVER(9)

NAME
     driver - description of a device driver

SYNOPSIS
     #include <sys/param.h>
     #include <sys/device.h>
     #include <sys/errno.h>

     static int
     foo_match(device_t parent, cfdata_t match, void *aux);

     static void
     foo_attach(device_t parent, device_t self, void *aux);

     static int
     foo_detach(device_t self, int flags);

     static int
     foo_activate(device_t self, enum devact act);

     […]

     CFATTACH_DECL_NEW(foo,                    /* driver name */
             sizeof(struct foo_softc),         /* size of instance data */
             foo_match,                        /* match/probe function */
             foo_attach,                       /* attach function */
             foo_detach,                       /* detach function */
             foo_activate);                    /* activate function */
```


obsd64$ man driver
man: No entry for driver in the manual.

Short Case Study

umb(4): USB Mobile Broadband Interface Model (MBIM)

NAME

umb - USB Mobile Broadband Interface Model (MBIM)

SYNOPSIS

umb* at uhub?

DESCRIPTION

The umb driver provides support for USB MBIM devices.

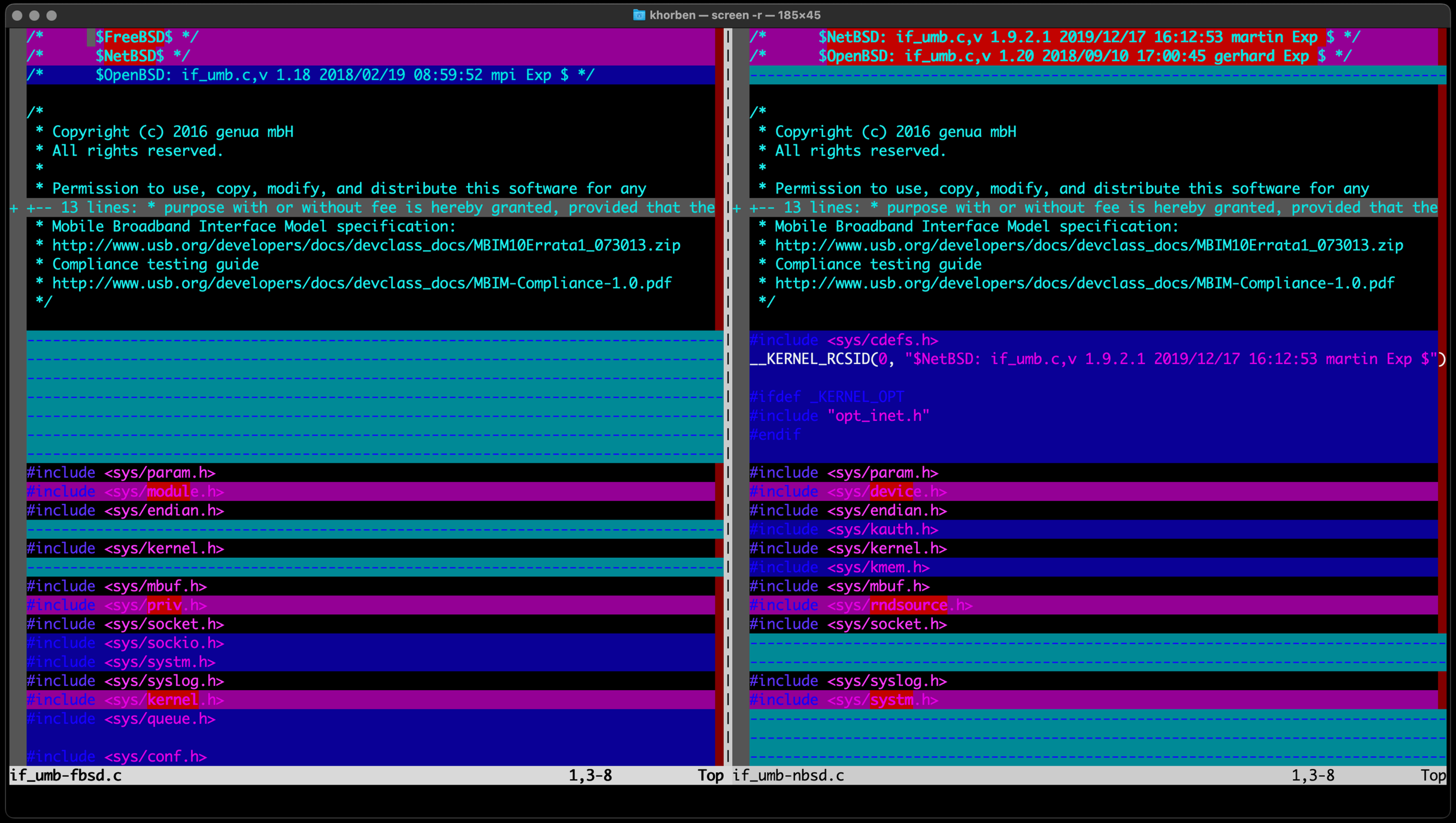
MBIM devices establish connections via cellular networks such as GPRS, UMTS, and LTE. They appear as a regular point-to-point network interface, transporting raw IP frames.

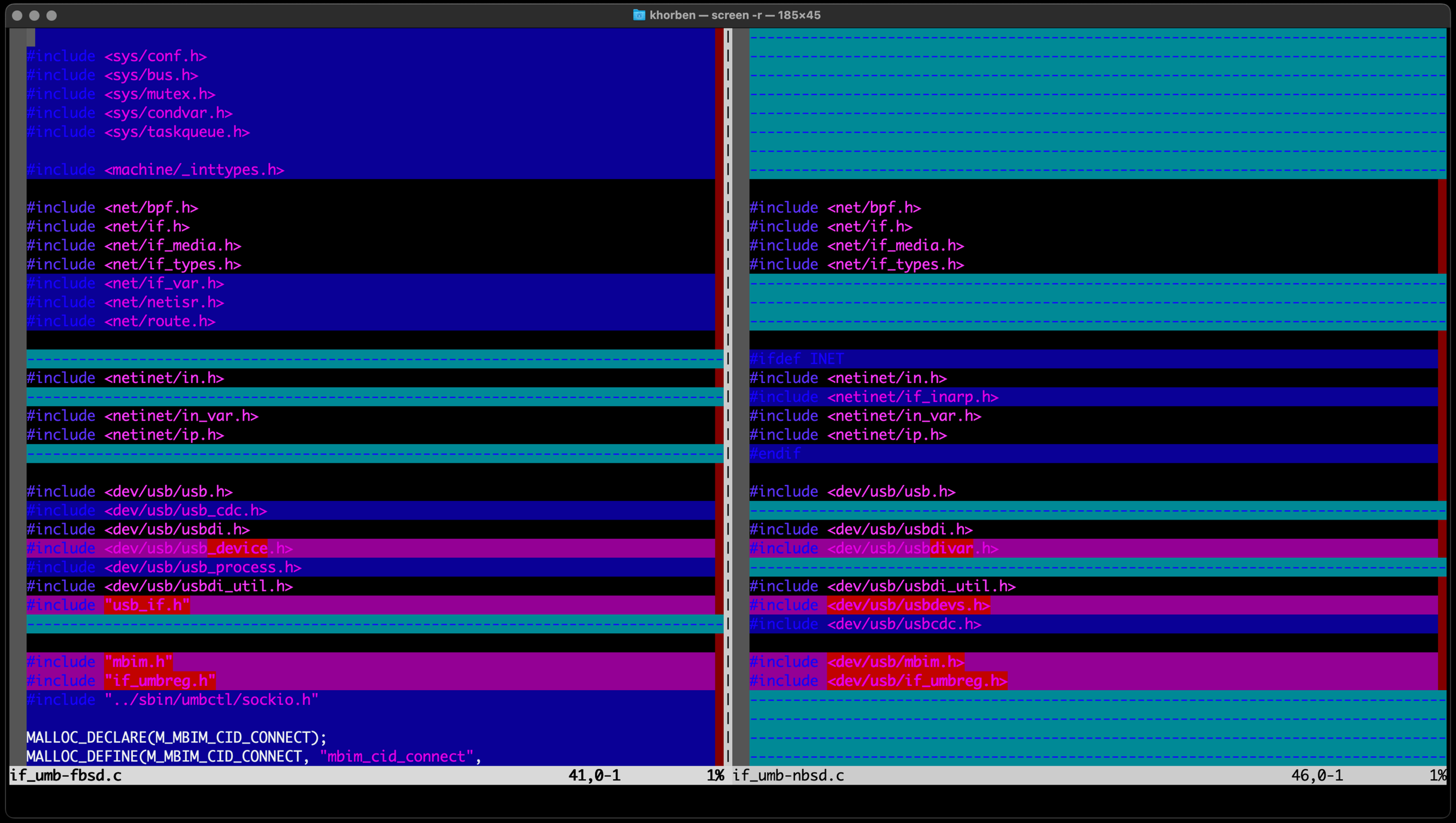
Required configuration parameters like PIN and APN have to be set with ifconfig(8). Once the SIM card has been unlocked with the correct PIN, it will remain in this state until the MBIM device is power-cycled. In case the device is connected to an "always-on" USB port, it may be possible to connect to a provider without entering the PIN again even if the system was rebooted.

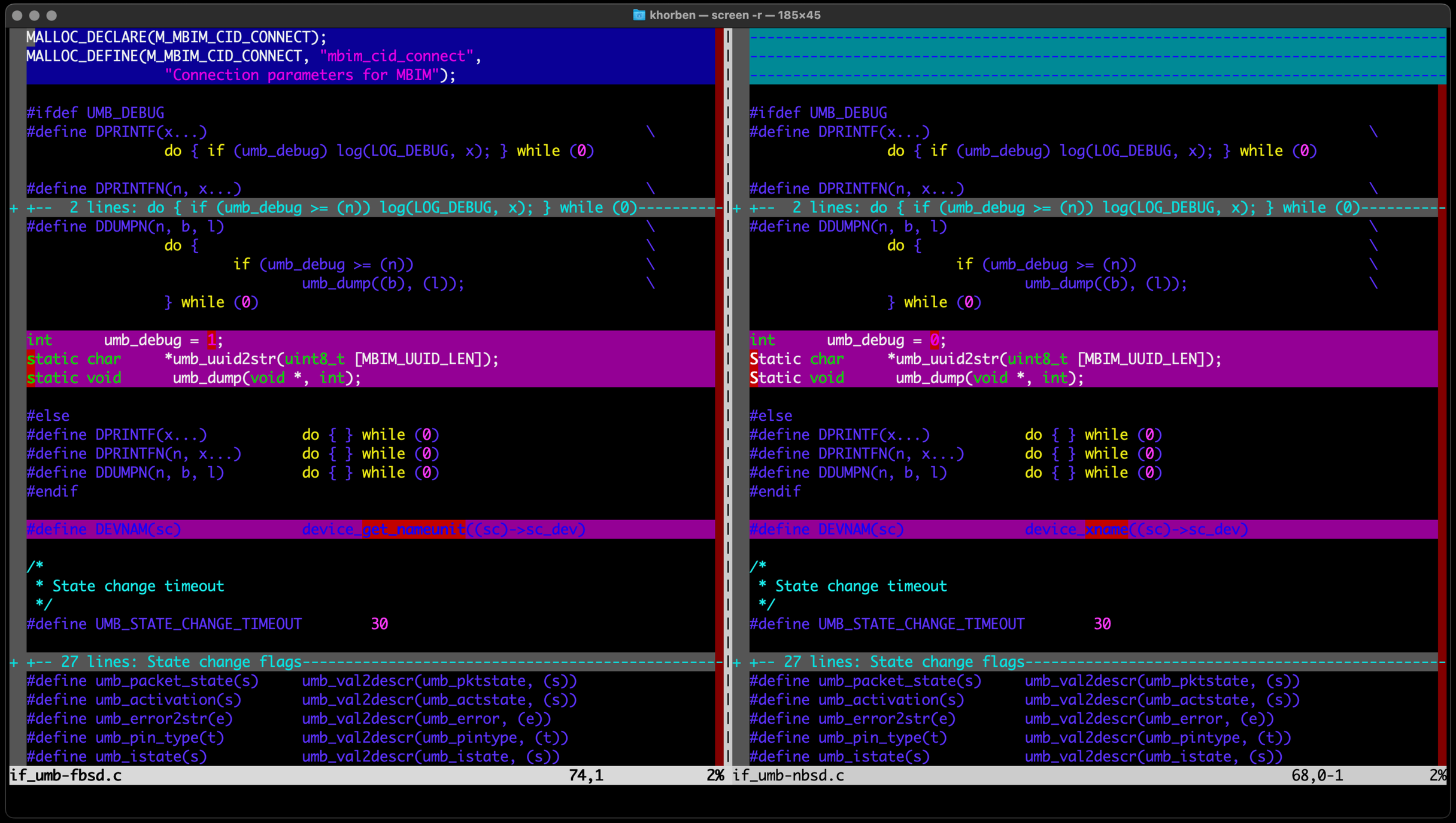
HARDWARE

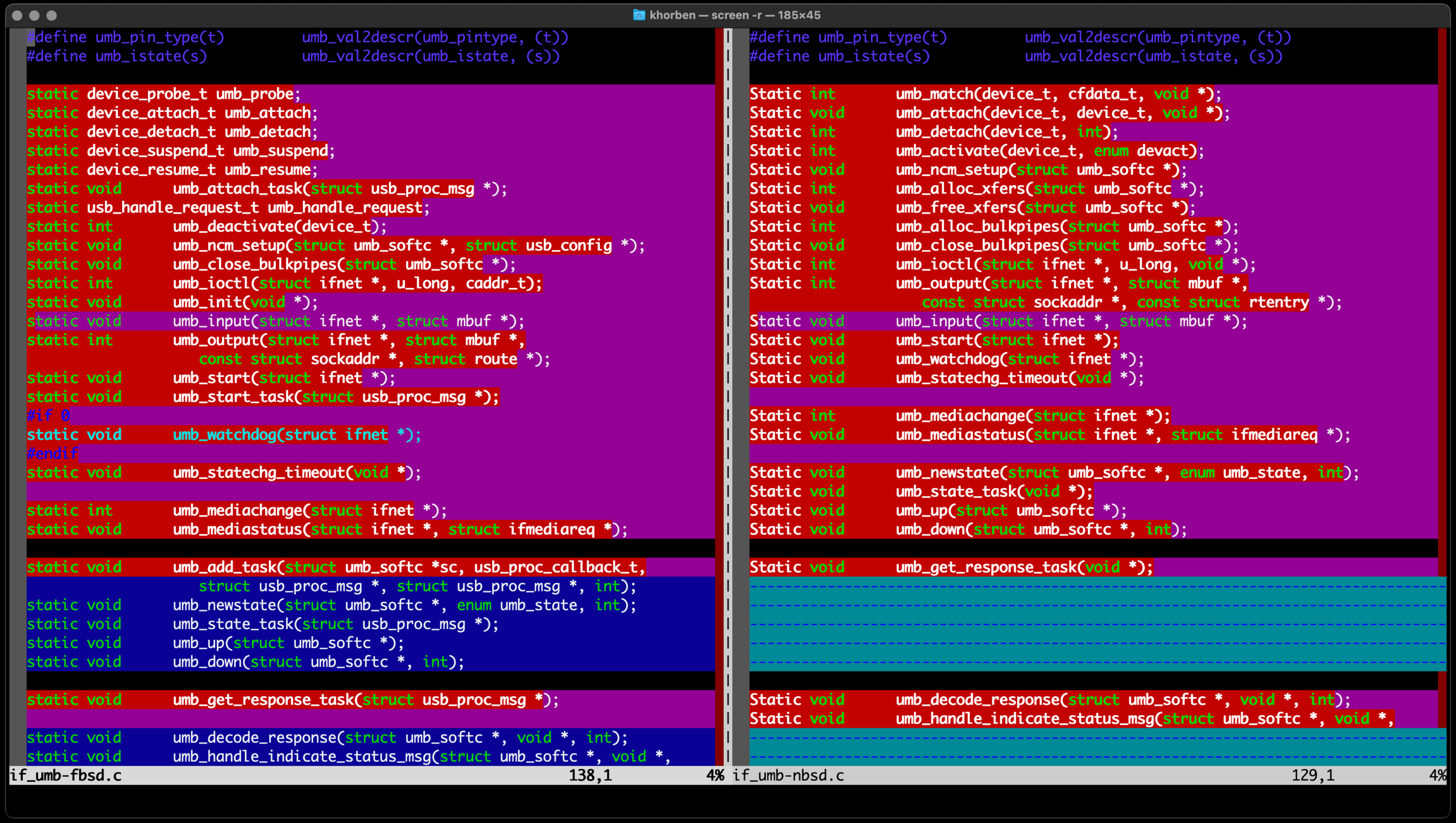
The following devices should work:

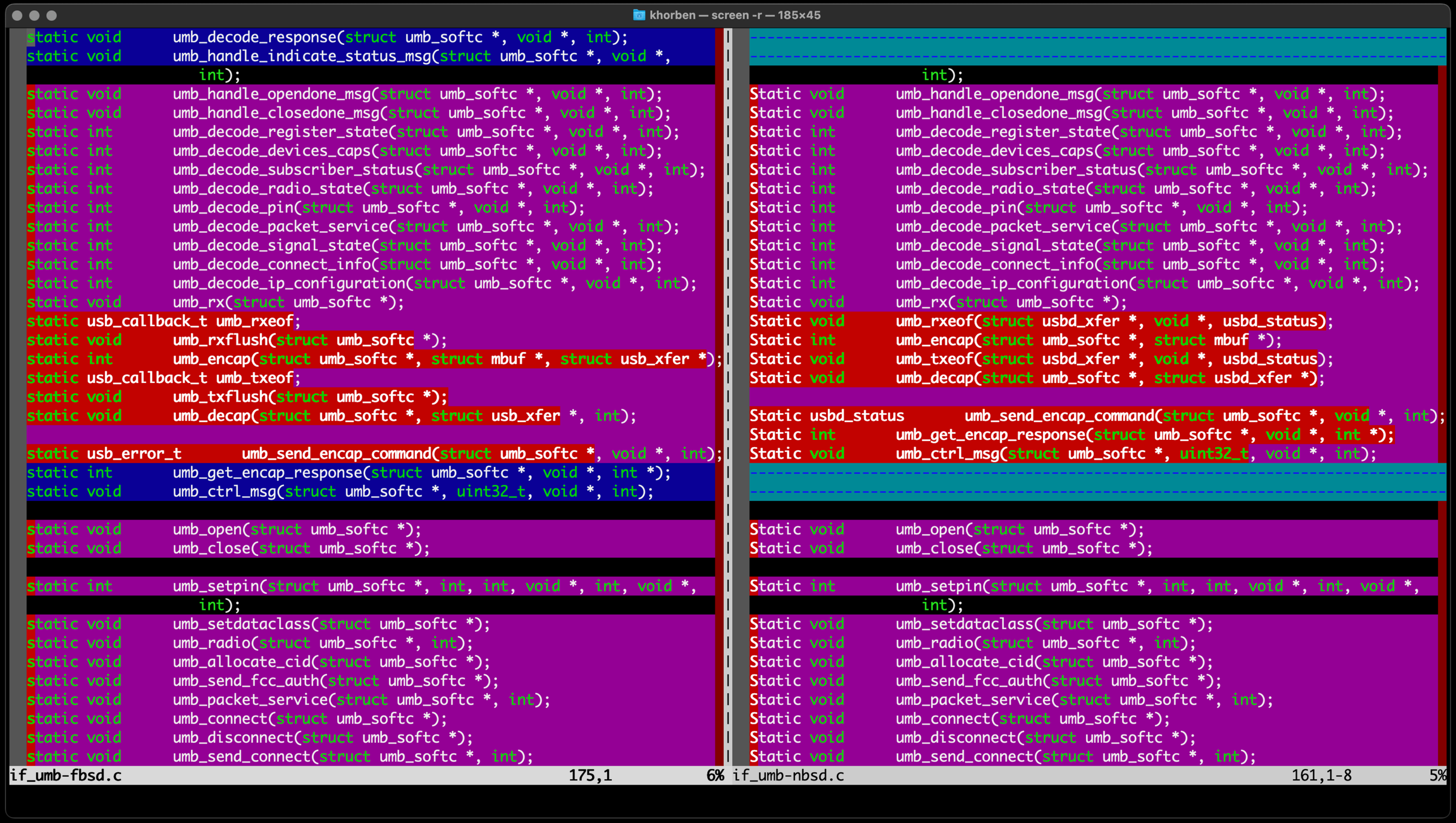
- Dell DW5821e
- Ericsson H5321gw and N5321gw
- Fibocom L831-EAU
- Medion Mobile S4222 (MediaTek OEM)
- Quectel EC25
- SIMCom SIM7600
- Sierra Wireless EM7345
- Sierra Wireless EM7455
- Sierra Wireless EM8805
- Sierra Wireless MC8305

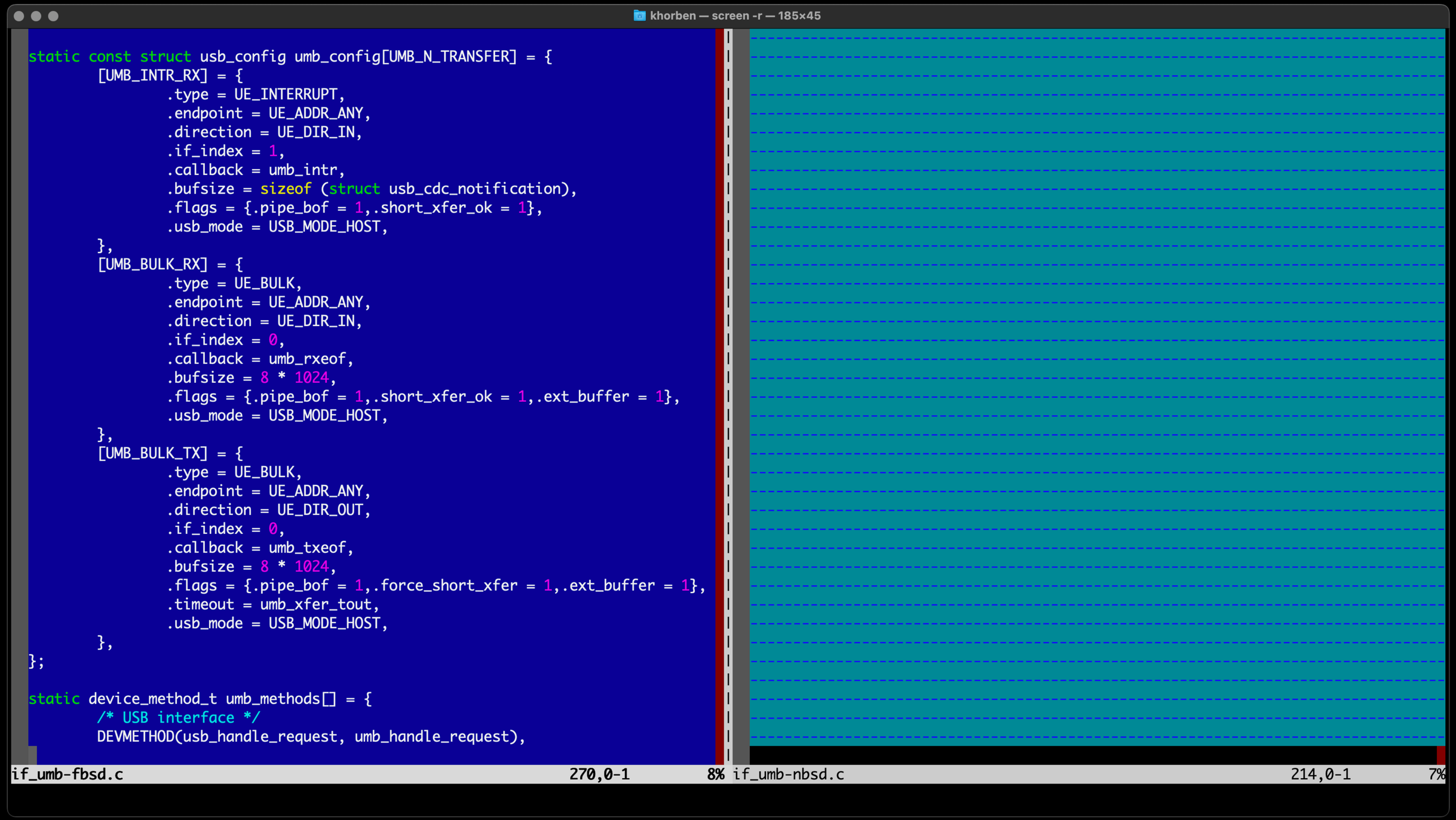

khorben — screen -r — 185×45
/*	$FreeBSD$ */
/*	$NetBSD$ */
/*	$OpenBSD: if_umb.c,v 1.18 2018/02/19 08:59:52 mpi Exp $ */
/*
* Copyright (c) 2016 genua mbH
* All rights reserved.
*
* Permission to use, copy, modify, and distribute this software for any
+ +-- 13 lines: * purpose with or without fee is hereby granted, provided that the
* Mobile Broadband Interface Model specification:
* http://www.usb.org/developers/docs/devclass_docs/MBIM10Errata1_073013.zip
* Compliance testing guide
* http://www.usb.org/developers/docs/devclass_docs/MBIM-Compliance-1.0.pdf
*/
#include <sys/param.h>
#include <sys/module.h>
#include <sys/endian.h>
#include <sys/kernel.h>
#include <sys/mbuf.h>
#include <sys/priv.h>
#include <sys/socket.h>
#include <sys/sockio.h>
#include <sys/systm.h>
#include <sys/syslog.h>
#include <sys/kernel.h>
#include <sys/queue.h>
#include <sys/conf.h>
if_umb-fbsd.c 1,3-8 Top
/*	$NetBSD: if_umb.c,v 1.9.2.1 2019/12/17 16:12:53 martin Exp $ */
/*	$OpenBSD: if_umb.c,v 1.20 2018/09/10 17:00:45 gerhard Exp $ */
/*
* Copyright (c) 2016 genua mbH
* All rights reserved.
*
* Permission to use, copy, modify, and distribute this software for any
+ +-- 13 lines: * purpose with or without fee is hereby granted, provided that the
* Mobile Broadband Interface Model specification:
* http://www.usb.org/developers/docs/devclass_docs/MBIM10Errata1_073013.zip
* Compliance testing guide
* http://www.usb.org/developers/docs/devclass_docs/MBIM-Compliance-1.0.pdf
*/
#include <sys/cdefs.h>
__KERNEL_RCSID(0, "$NetBSD: if_umb.c,v 1.9.2.1 2019/12/17 16:12:53 martin Exp $")
#ifdef _KERNEL_OPT
#include "opt_inet.h"
#endif
#include <sys/param.h>
#include <sys/device.h>
#include <sys/endian.h>
#include <sys/kauth.h>
#include <sys/kernel.h>
#include <sys/kmem.h>
#include <sys/mbuf.h>
#include <sys/rndsource.h>
#include <sys/socket.h>
#include <sys/syslog.h>
#include <sys/systm.h>
if_umb-nbsd.c 1,3-8 Top

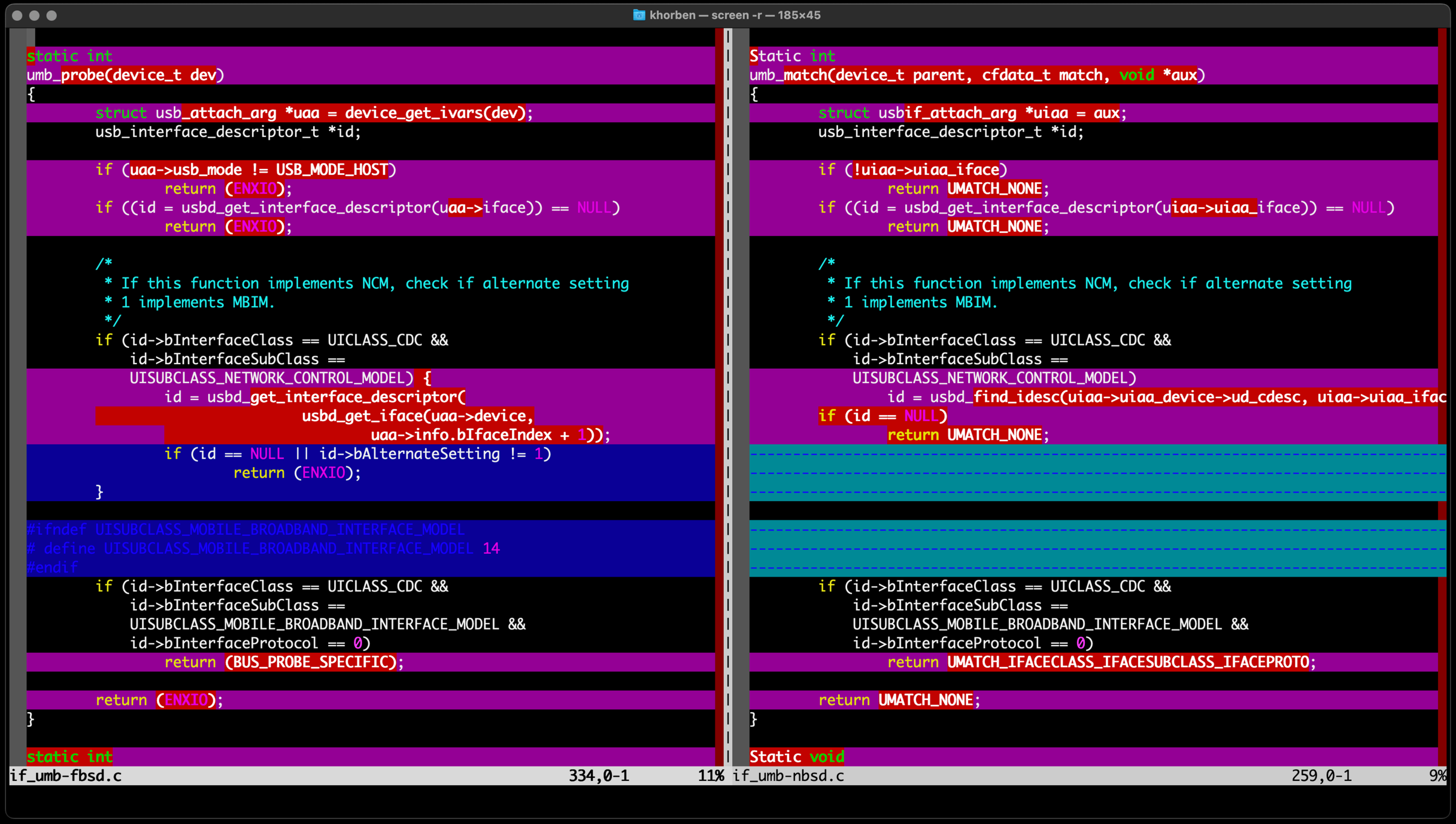

khorben — screen -r — 185×45
#include <sys/conf.h>
#include <sys/bus.h>
#include <sys/mutex.h>
#include <sys/condvar.h>
#include <sys/taskqueue.h>
#include <machine/_inttypes.h>
#include <net/bpf.h>
#include <net/if.h>
#include <net/if_media.h>
#include <net/if_types.h>
#include <net/if_var.h>
#include <net/netisr.h>
#include <net/route.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <netinet/ip.h>
#include <dev/usb/usb.h>
#include <dev/usb/usb_cdc.h>
#include <dev/usb/usbdi.h>
#include <dev/usb/usb_device.h>
#include <dev/usb/usb_process.h>
#include <dev/usb/usbdi_util.h>
#include "usb_if.h"
#include "mbim.h"
#include "if_umbreg.h"
#include "../sbin/umbctl/sockio.h"
MALLOC_DECLARE(M_MBIM_CID_CONNECT);
MALLOC_DEFINE(M_MBIM_CID_CONNECT, "mbim_cid_connect",
if_umb-fbsd.c 41,0-1 1%
#include <net/bpf.h>
#include <net/if.h>
#include <net/if_media.h>
#include <net/if_types.h>
#ifdef INET
#include <netinet/in.h>
#include <netinet/if_inarp.h>
#include <netinet/in_var.h>
#include <netinet/ip.h>
#endif
#include <dev/usb/usb.h>
#include <dev/usb/usbdi.h>
#include <dev/usb/usbdivar.h>
#include <dev/usb/usbdi_util.h>
#include <dev/usb/usbdevs.h>
#include <dev/usb/usbcdc.h>
#include <dev/usb/mbim.h>
#include <dev/usb/if_umbreg.h>
if_umb-nbsd.c 46,0-1 1%

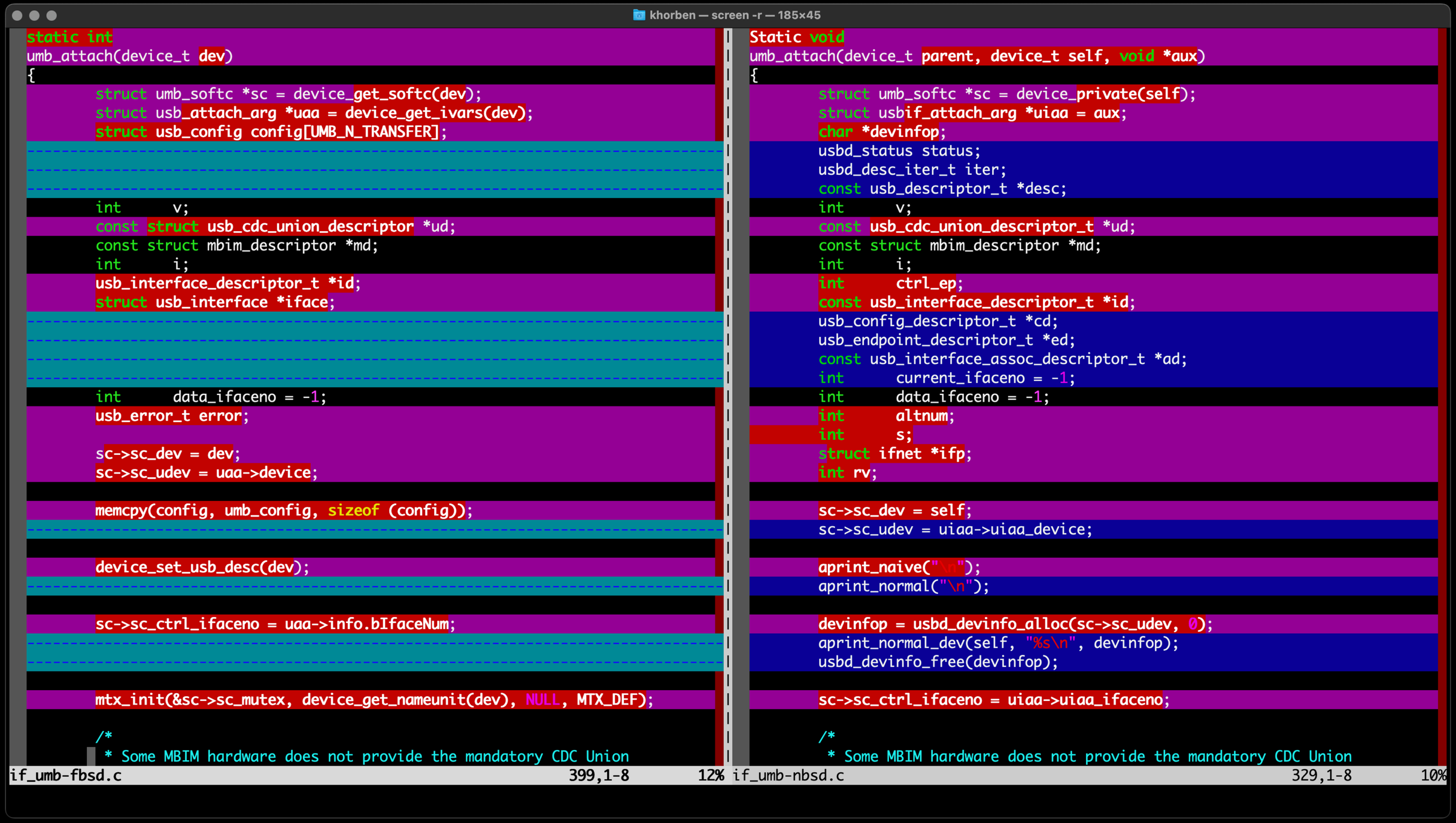

khorben — screen -r — 185×45
MALLOC_DECLARE(M_MBIM_CID_CONNECT);
MALLOC_DEFINE(M_MBIM_CID_CONNECT, "mbim_cid_connect",
"Connection parameters for MBIM");
#ifdef UMB_DEBUG
#define DPRINTF(x...) \
do { if (umb_debug) log(LOG_DEBUG, x); } while (0)
#define DPRINTFN(n, x...) \
+-- 2 lines: do { if (umb_debug >= (n)) log(LOG_DEBUG, x); } while (0)----------
#define DDUMPN(n, b, l) \
do { \
if (umb_debug >= (n)) \
umb_dump((b), (l)); \
} while (0)
int umb_debug = 1;
static char *umb_uuid2str(uint8_t [MBIM_UUID_LEN]);
static void umb_dump(void *, int);
#else
#define DPRINTF(x...) do { } while (0)
#define DPRINTFN(n, x...) do { } while (0)
#define DDUMPN(n, b, l) do { } while (0)
#endif
#define DEVNAM(sc) device_get_nameunit((sc)->sc_dev)
/*
* State change timeout
*/
#define UMB_STATE_CHANGE_TIMEOUT 30
+-- 27 lines: State change flags----------
#define umb_packet_state(s) umb_val2descr(umb_pktstate, (s))
#define umb_activation(s) umb_val2descr(umb_actstate, (s))
#define umb_error2str(e) umb_val2descr(umb_error, (e))
#define umb_pin_type(t) umb_val2descr(umb_pintype, (t))
#define umb_istate(s) umb_val2descr(umb_istate, (s))
if_umb-fbsd.c 74,1 2%
#ifdef UMB_DEBUG
#define DPRINTF(x...) \
do { if (umb_debug) log(LOG_DEBUG, x); } while (0)
#define DPRINTFN(n, x...) \
+-- 2 lines: do { if (umb_debug >= (n)) log(LOG_DEBUG, x); } while (0)----------
#define DDUMPN(n, b, l) \
do { \
if (umb_debug >= (n)) \
umb_dump((b), (l)); \
} while (0)
int umb_debug = 0;
Static char *umb_uuid2str(uint8_t [MBIM_UUID_LEN]);
Static void umb_dump(void *, int);
#else
#define DPRINTF(x...) do { } while (0)
#define DPRINTFN(n, x...) do { } while (0)
#define DDUMPN(n, b, l) do { } while (0)
#endif
#define DEVNAM(sc) device_xname((sc)->sc_dev)
/*
* State change timeout
*/
#define UMB_STATE_CHANGE_TIMEOUT 30
+-- 27 lines: State change flags----------
#define umb_packet_state(s) umb_val2descr(umb_pktstate, (s))
#define umb_activation(s) umb_val2descr(umb_actstate, (s))
#define umb_error2str(e) umb_val2descr(umb_error, (e))
#define umb_pin_type(t) umb_val2descr(umb_pintype, (t))
#define umb_istate(s) umb_val2descr(umb_istate, (s))
if_umb-nbsd.c 68,0-1 2%

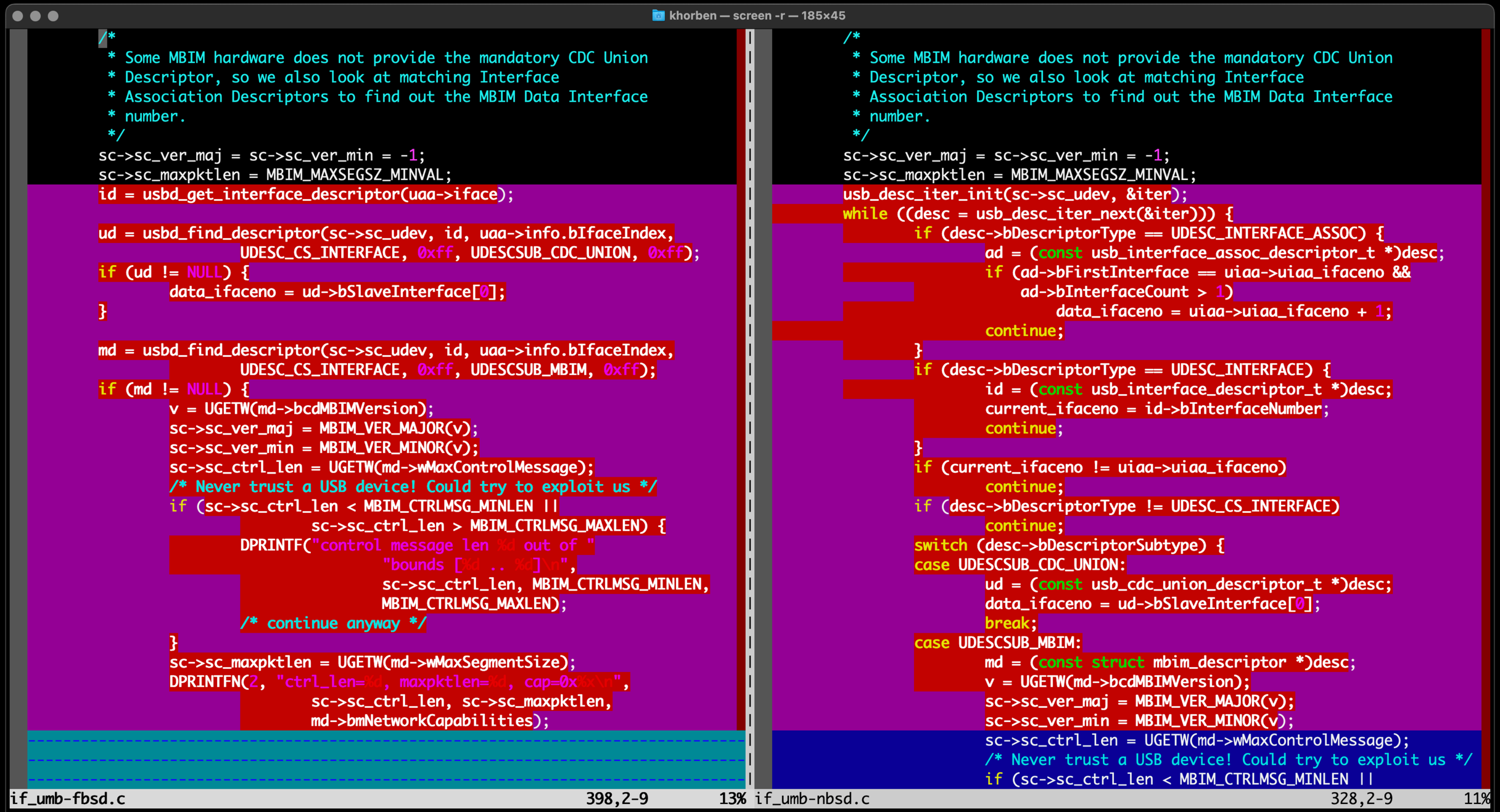

khorben — screen -r — 185×45
#define umb_pin_type(t) umb_val2descr(umb_pintype, (t))
#define umb_istate(s) umb_val2descr(umb_istate, (s))
static device_probe_t umb_probe;
static device_attach_t umb_attach;
static device_detach_t umb_detach;
static device_suspend_t umb_suspend;
static device_resume_t umb_resume;
static void umb_attach_task(struct usb_proc_msg *);
static usb_handle_request_t umb_handle_request;
static int umb_deactivate(device_t);
static void umb_ncm_setup(struct umb_softc *, struct usb_config *);
static void umb_close_bulkpipes(struct umb_softc *);
static int umb_ioctl(struct ifnet *, u_long, caddr_t);
static void umb_init(void *);
static void umb_input(struct ifnet *, struct mbuf *);
static int umb_output(struct ifnet *, struct mbuf *,
const struct sockaddr *, struct route *);
static void umb_start(struct ifnet *);
static void umb_start_task(struct usb_proc_msg *);
#if 0
static void umb_watchdog(struct ifnet *);
#endif
static void umb_statechg_timeout(void *);
static int umb_mediachange(struct ifnet *);
static void umb_mediastatus(struct ifnet *, struct ifmediareq *);
static void umb_add_task(struct umb_softc *sc, usb_proc_callback_t,
struct usb_proc_msg *, struct usb_proc_msg *, int);
static void umb_newstate(struct umb_softc *, enum umb_state, int);
static void umb_state_task(struct usb_proc_msg *);
static void umb_up(struct umb_softc *);
static void umb_down(struct umb_softc *, int);
static void umb_get_response_task(struct usb_proc_msg *);
static void umb_decode_response(struct umb_softc *, void *, int);
static void umb_handle_indicate_status_msg(struct umb_softc *, void *,
if_umb-fbsd.c 138,1 4%
#define umb_pin_type(t) umb_val2descr(umb_pintype, (t))
#define umb_istate(s) umb_val2descr(umb_istate, (s))
Static int umb_match(device_t, cfdata_t, void *);
Static void umb_attach(device_t, device_t, void *);
Static int umb_detach(device_t, int);
Static int umb_activate(device_t, enum devact);
Static void umb_ncm_setup(struct umb_softc *);
Static int umb_alloc_xfers(struct umb_softc *);
Static void umb_free_xfers(struct umb_softc *);
Static int umb_alloc_bulkpipes(struct umb_softc *);
Static void umb_close_bulkpipes(struct umb_softc *);
Static int umb_ioctl(struct ifnet *, u_long, void *);
Static int umb_output(struct ifnet *, struct mbuf *,
const struct sockaddr *, const struct rtentry *);
Static void umb_input(struct ifnet *, struct mbuf *);
Static void umb_start(struct ifnet *);
Static void umb_watchdog(struct ifnet *);
Static void umb_statechg_timeout(void *);
Static int umb_mediachange(struct ifnet *);
Static void umb_mediastatus(struct ifnet *, struct ifmediareq *);
Static void umb_newstate(struct umb_softc *, enum umb_state, int);
Static void umb_state_task(void *);
Static void umb_up(struct umb_softc *);
Static void umb_down(struct umb_softc *, int);
Static void umb_get_response_task(void *);
Static void umb_decode_response(struct umb_softc *, void *, int);
Static void umb_handle_indicate_status_msg(struct umb_softc *, void *,
if_umb-nbsd.c 129,1 4%

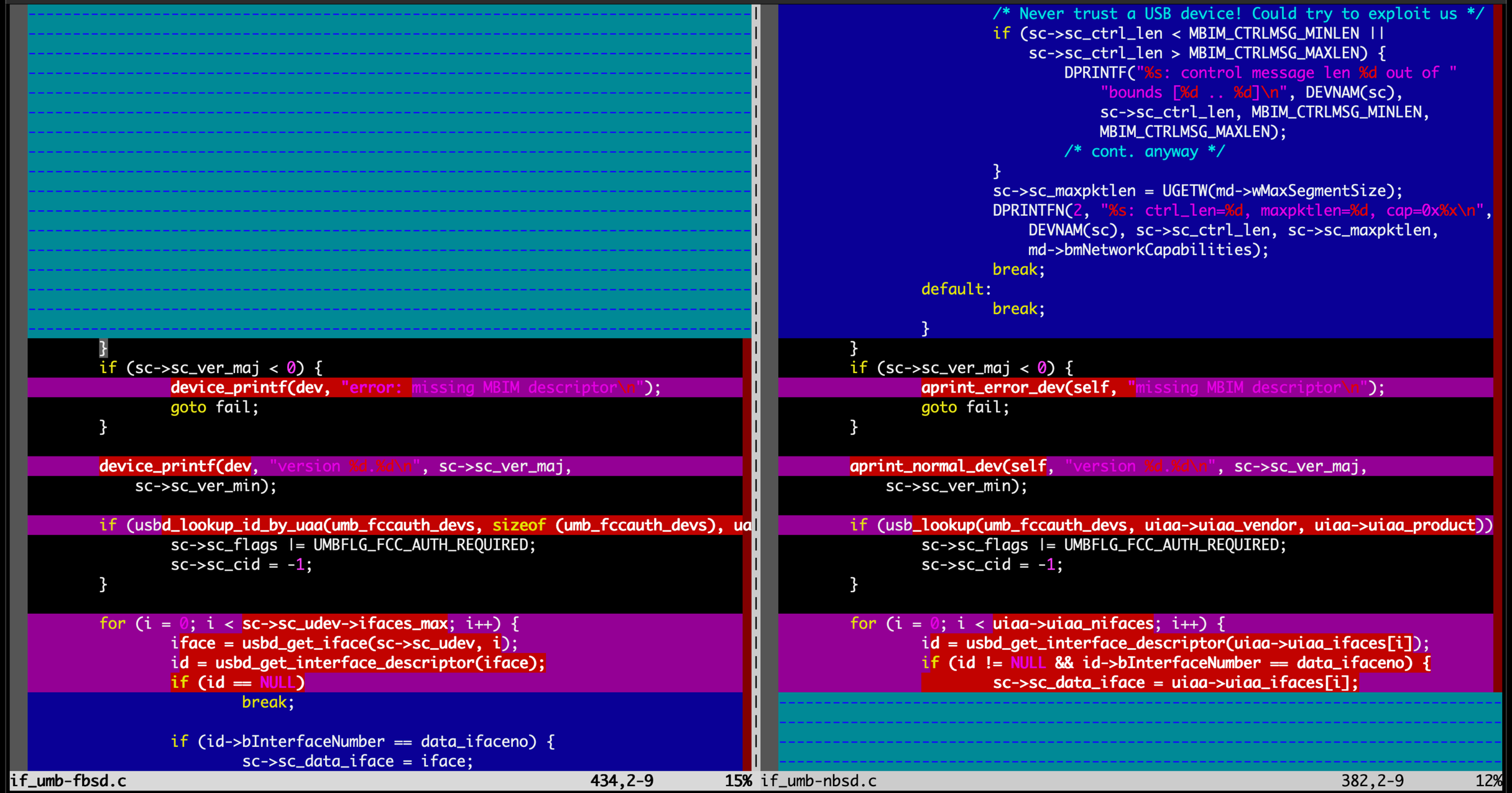

khorben — screen -r — 185×45
static void umb_decode_response(struct umb_softc *, void *, int);
static void umb_handle_indicate_status_msg(struct umb_softc *, void *,
int);
static void umb_handle_opendone_msg(struct umb_softc *, void *, int);
static void umb_handle_closedone_msg(struct umb_softc *, void *, int);
static int umb_decode_register_state(struct umb_softc *, void *, int);
static int umb_decode_devices_caps(struct umb_softc *, void *, int);
static int umb_decode_subscriber_status(struct umb_softc *, void *, int);
static int umb_decode_radio_state(struct umb_softc *, void *, int);
static int umb_decode_pin(struct umb_softc *, void *, int);
static int umb_decode_packet_service(struct umb_softc *, void *, int);
static int umb_decode_signal_state(struct umb_softc *, void *, int);
static int umb_decode_connect_info(struct umb_softc *, void *, int);
static int umb_decode_ip_configuration(struct umb_softc *, void *, int);
static void umb_rx(struct umb_softc *);
static usb_callback_t umb_rxeof;
static void umb_rxflush(struct umb_softc *);
static int umb_encap(struct umb_softc *, struct mbuf *, struct usb_xfer *);
static usb_callback_t umb_txeof;
static void umb_txflush(struct umb_softc *);
static void umb_decap(struct umb_softc *, struct usb_xfer *, int);
static usb_error_t umb_send_encap_command(struct umb_softc *, void *, int);
static int umb_get_encap_response(struct umb_softc *, void *, int *);
static void umb_ctrl_msg(struct umb_softc *, uint32_t, void *, int);
static void umb_open(struct umb_softc *);
static void umb_close(struct umb_softc *);
static int umb_setpin(struct umb_softc *, int, int, void *, int, void *,
int);
static void umb_setdataclass(struct umb_softc *);
static void umb_radio(struct umb_softc *, int);
static void umb_allocate_cid(struct umb_softc *);
static void umb_send_fcc_auth(struct umb_softc *);
static void umb_packet_service(struct umb_softc *, int);
static void umb_connect(struct umb_softc *);
static void umb_disconnect(struct umb_softc *);
static void umb_send_connect(struct umb_softc *, int);
int);
Static void umb_handle_opendone_msg(struct umb_softc *, void *, int);
Static void umb_handle_closedone_msg(struct umb_softc *, void *, int);
Static int umb_decode_register_state(struct umb_softc *, void *, int);
Static int umb_decode_devices_caps(struct umb_softc *, void *, int);
Static int umb_decode_subscriber_status(struct umb_softc *, void *, int);
Static int umb_decode_radio_state(struct umb_softc *, void *, int);
Static int umb_decode_pin(struct umb_softc *, void *, int);
Static int umb_decode_packet_service(struct umb_softc *, void *, int);
Static int umb_decode_signal_state(struct umb_softc *, void *, int);
Static int umb_decode_connect_info(struct umb_softc *, void *, int);
Static int umb_decode_ip_configuration(struct umb_softc *, void *, int);
Static void umb_rx(struct umb_softc *);
Static void umb_rxeof(struct usbd_xfer *, void *, usbd_status);
Static int umb_encap(struct umb_softc *, struct mbuf *);
Static void umb_txeof(struct usbd_xfer *, void *, usbd_status);
Static void umb_decap(struct umb_softc *, struct usbd_xfer *);
Static usbd_status umb_send_encap_command(struct umb_softc *, void *, int);
Static int umb_get_encap_response(struct umb_softc *, void *, int *);
Static void umb_ctrl_msg(struct umb_softc *, uint32_t, void *, int);
Static void umb_open(struct umb_softc *);
Static void umb_close(struct umb_softc *);
Static int umb_setpin(struct umb_softc *, int, int, void *, int, void *,
int);
Static void umb_setdataclass(struct umb_softc *);
Static void umb_radio(struct umb_softc *, int);
Static void umb_allocate_cid(struct umb_softc *);
Static void umb_send_fcc_auth(struct umb_softc *);
Static void umb_packet_service(struct umb_softc *, int);
Static void umb_connect(struct umb_softc *);
Static void umb_disconnect(struct umb_softc *);
Static void umb_send_connect(struct umb_softc *, int);
if_umb-fbsd.c 175,1 6%
if_umb-nbsd.c 161,1-8 5%

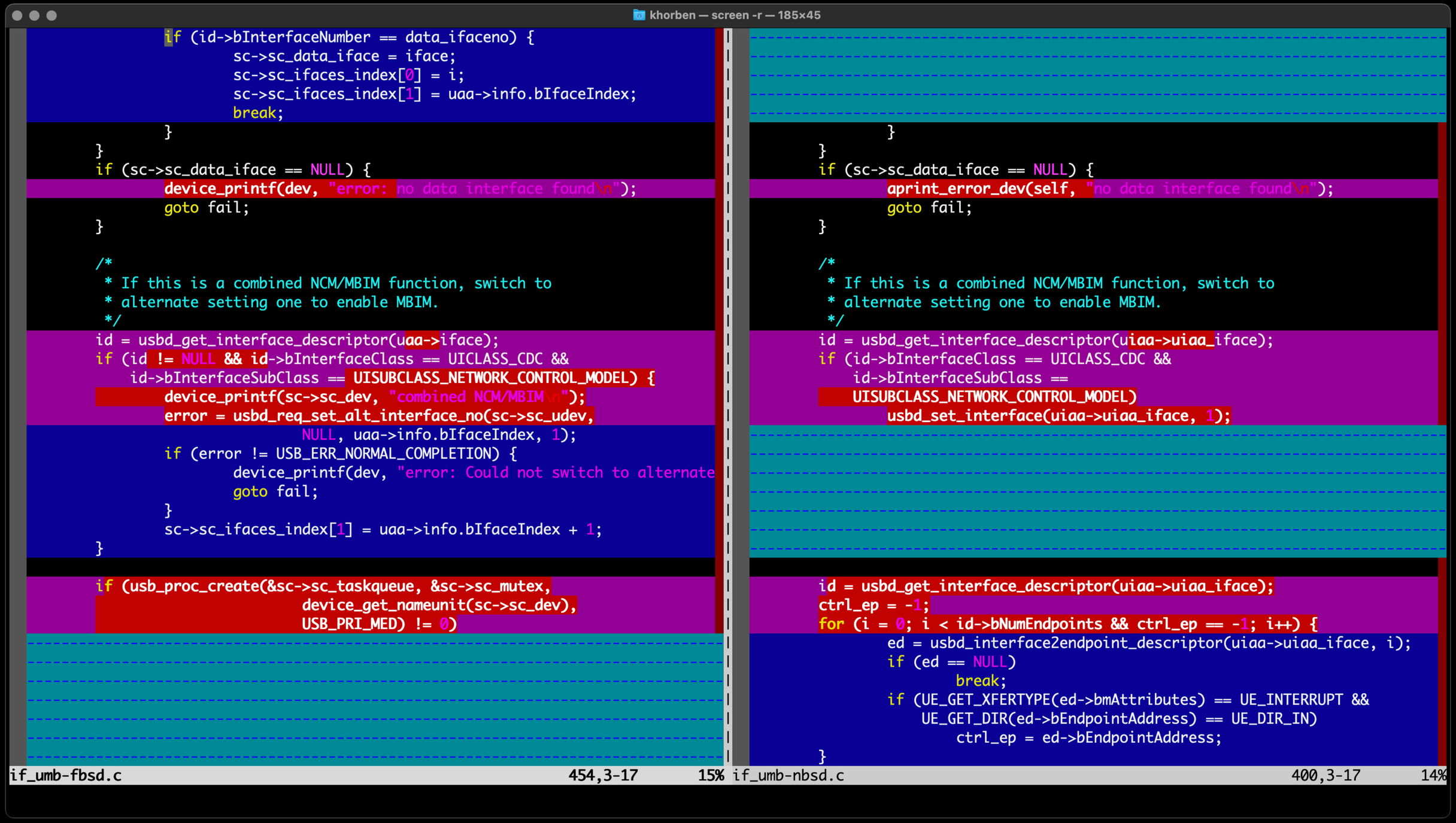

khorben — screen -r — 185×45
static const struct usb_config umb_config[UMB_N_TRANSFER] = {
[UMB_INTR_RX] = {
.type = UE_INTERRUPT,
.endpoint = UE_ADDR_ANY,
.direction = UE_DIR_IN,
.if_index = 1,
.callback = umb_intr,
.bufsize = sizeof (struct usb_cdc_notification),
.flags = {.pipe_bof = 1,.short_xfer_ok = 1},
.usb_mode = USB_MODE_HOST,
},
[UMB_BULK_RX] = {
.type = UE_BULK,
.endpoint = UE_ADDR_ANY,
.direction = UE_DIR_IN,
.if_index = 0,
.callback = umb_rxeof,
.bufsize = 8 * 1024,
.flags = {.pipe_bof = 1,.short_xfer_ok = 1,.ext_buffer = 1},
.usb_mode = USB_MODE_HOST,
},
[UMB_BULK_TX] = {
.type = UE_BULK,
.endpoint = UE_ADDR_ANY,
.direction = UE_DIR_OUT,
.if_index = 0,
.callback = umb_txeof,
.bufsize = 8 * 1024,
.flags = {.pipe_bof = 1,.force_short_xfer = 1,.ext_buffer = 1},
.timeout = umb_xfer_tout,
.usb_mode = USB_MODE_HOST,
},
};
static device_method_t umb_methods[] = {
/* USB interface */
DEVMETHOD(usb_handle_request, umb_handle_request),
if_umb-fbsd.c 270,0-1 8%
if_umb-nbsd.c 214,0-1 7%

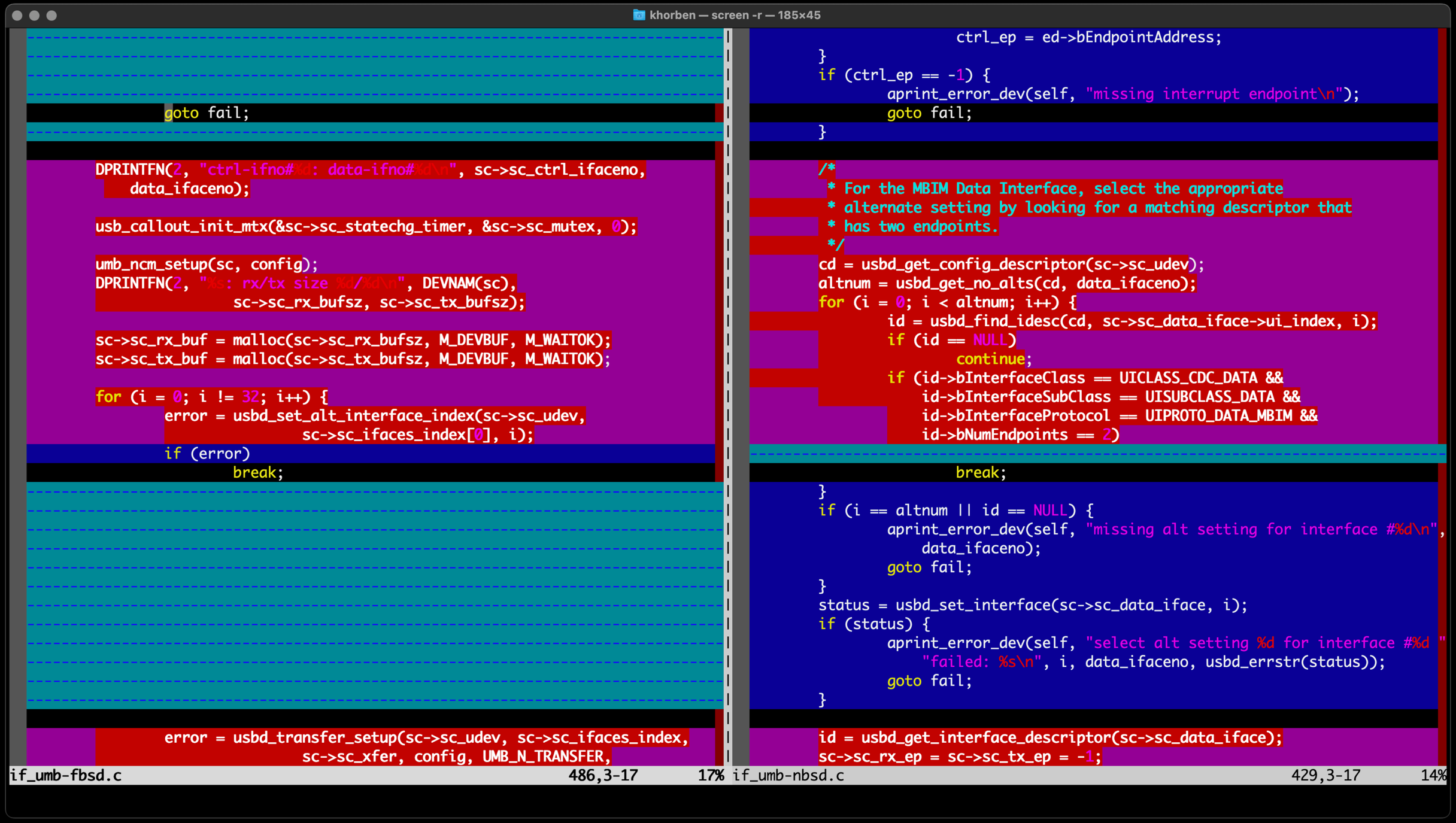

khorben — screen -r — 185×45
static int
umb_probe(device_t dev)
{
struct usb_attach_arg *uaa = device_get_ivars(dev);
usb_interface_descriptor_t *id;
if (uaa->usb_mode != USB_MODE_HOST)
return (ENXIO);
if ((id = usbd_get_interface_descriptor(uaa->iface)) == NULL)
return (ENXIO);
/*
* If this function implements NCM, check if alternate setting
* 1 implements MBIM.
*/
if (id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass ==
UISUBCLASS_NETWORK_CONTROL_MODEL) {
id = usbd_get_interface_descriptor(
usbd_get_iface(uaa->device,
uaa->info.bIfaceIndex + 1));
if (id == NULL || id->bAlternateSetting != 1)
return (ENXIO);
}
#ifndef UISUBCLASS_MOBILE_BROADBAND_INTERFACE_MODEL
define UISUBCLASS_MOBILE_BROADBAND_INTERFACE_MODEL 14
#endif
if (id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass ==
UISUBCLASS_MOBILE_BROADBAND_INTERFACE_MODEL &&
id->bInterfaceProtocol == 0)
return (BUS_PROBE_SPECIFIC);
return (ENXIO);
}
static int
if_umb-fbsd.c 334,0-1 11%
Static int
umb_match(device_t parent, cfdata_t match, void *aux)
{
struct usbif_attach_arg *uiaa = aux;
usb_interface_descriptor_t *id;
if (!uiaa->uiaa_iface)
return UMATCH_NONE;
if ((id = usbd_get_interface_descriptor(uiaa->uiaa_iface)) == NULL)
return UMATCH_NONE;
/*
* If this function implements NCM, check if alternate setting
* 1 implements MBIM.
*/
if (id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass ==
UISUBCLASS_NETWORK_CONTROL_MODEL)
id = usbd_find_idesc(uiaa->uiaa_device->ud_cdesc, uiaa->uiaa_ifac
if (id == NULL)
return UMATCH_NONE;
if (id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass ==
UISUBCLASS_MOBILE_BROADBAND_INTERFACE_MODEL &&
id->bInterfaceProtocol == 0)
return UMATCH_IFACECLASS_IFACESUBCLASS_IFACEPROTO;
return UMATCH_NONE;
}
Static void
if_umb-nbsd.c 259,0-1 9%

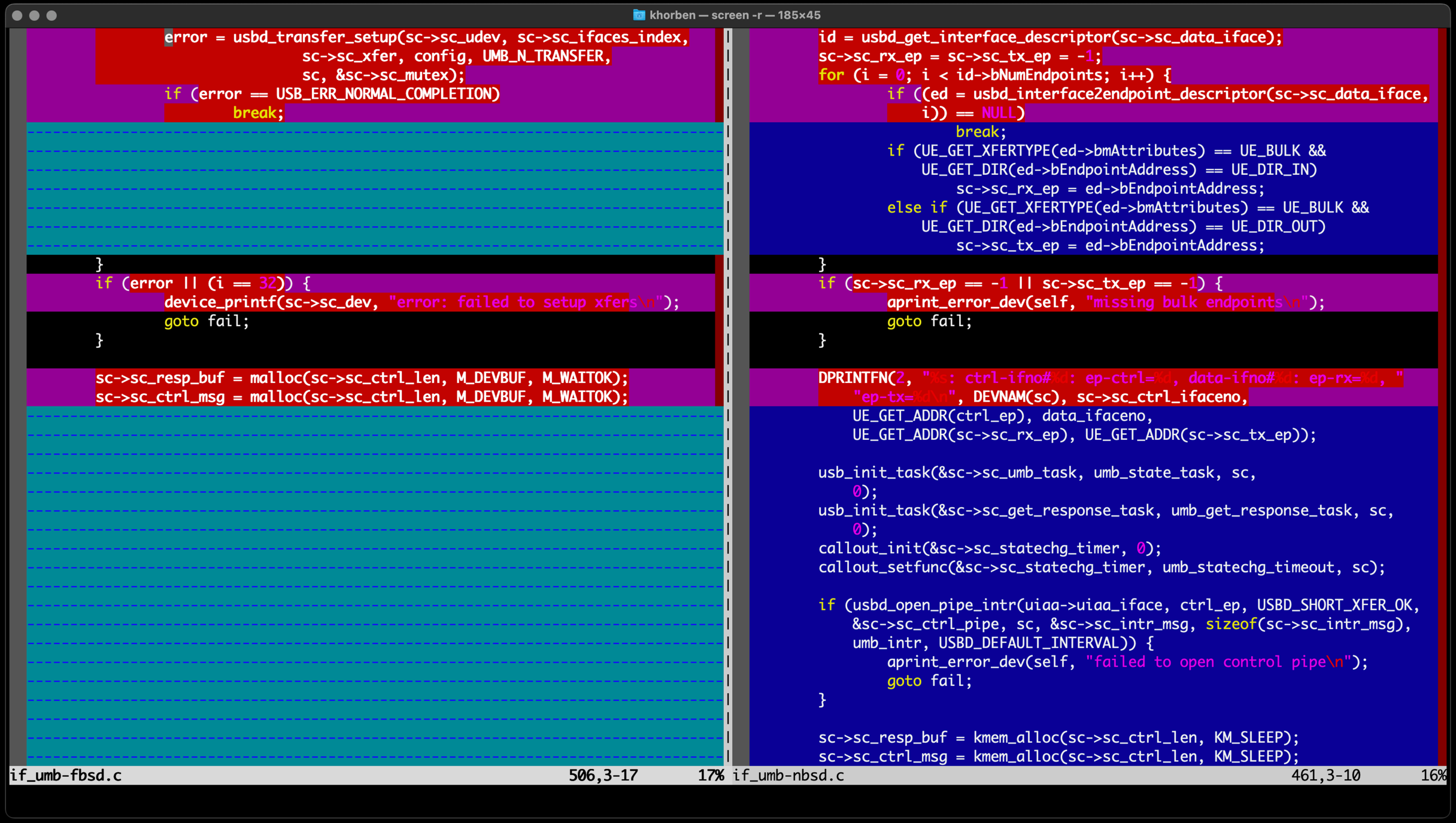

khorben — screen -r — 185×45
static int
umb_attach(device_t dev)
{
struct umb_softc *sc = device_get_softc(dev);
struct usb_attach_arg *uaa = device_get_ivars(dev);
struct usb_config config[UMB_N_TRANSFER];
int v;
const struct usb_cdc_union_descriptor *ud;
const struct mbim_descriptor *md;
int i;
usb_interface_descriptor_t *id;
struct usb_interface *iface;
int data_ifaceno = -1;
usb_error_t error;
sc->sc_dev = dev;
sc->sc_udev = uaa->device;
memcpy(config, umb_config, sizeof (config));
device_set_usb_desc(dev);
sc->sc_ctrl_ifaceno = uaa->info.bIfaceNum;
mtx_init(&sc->sc_mutex, device_get_nameunit(dev), NULL, MTX_DEF);
/*
* Some MBIM hardware does not provide the mandatory CDC Union
if_umb-fbsd.c 399,1-8 12%
Static void
umb_attach(device_t parent, device_t self, void *aux)
{
struct umb_softc *sc = device_private(self);
struct usbif_attach_arg *uiaa = aux;
char *devinfop;
usbd_status status;
usbd_desc_iter_t iter;
const usb_descriptor_t *desc;
int v;
const usb_cdc_union_descriptor_t *ud;
const struct mbim_descriptor *md;
int i;
int ctrl_ep;
const usb_interface_descriptor_t *id;
usb_config_descriptor_t *cd;
usb_endpoint_descriptor_t *ed;
const usb_interface_assoc_descriptor_t *ad;
int current_ifaceno = -1;
int data_ifaceno = -1;
int altnum;
int s;
struct ifnet *ifp;
int rv;
sc->sc_dev = self;
sc->sc_udev = uiaa->uiaa_device;
aprint_naive("\n");
aprint_normal("\n");
devinfop = usbd_devinfo_alloc(sc->sc_udev, 0);
aprint_normal_dev(self, "%s\n", devinfop);
usbd_devinfo_free(devinfop);
sc->sc_ctrl_ifaceno = uiaa->uiaa_ifaceno;
/*
* Some MBIM hardware does not provide the mandatory CDC Union
if_umb-nbsd.c 329,1-8 10%

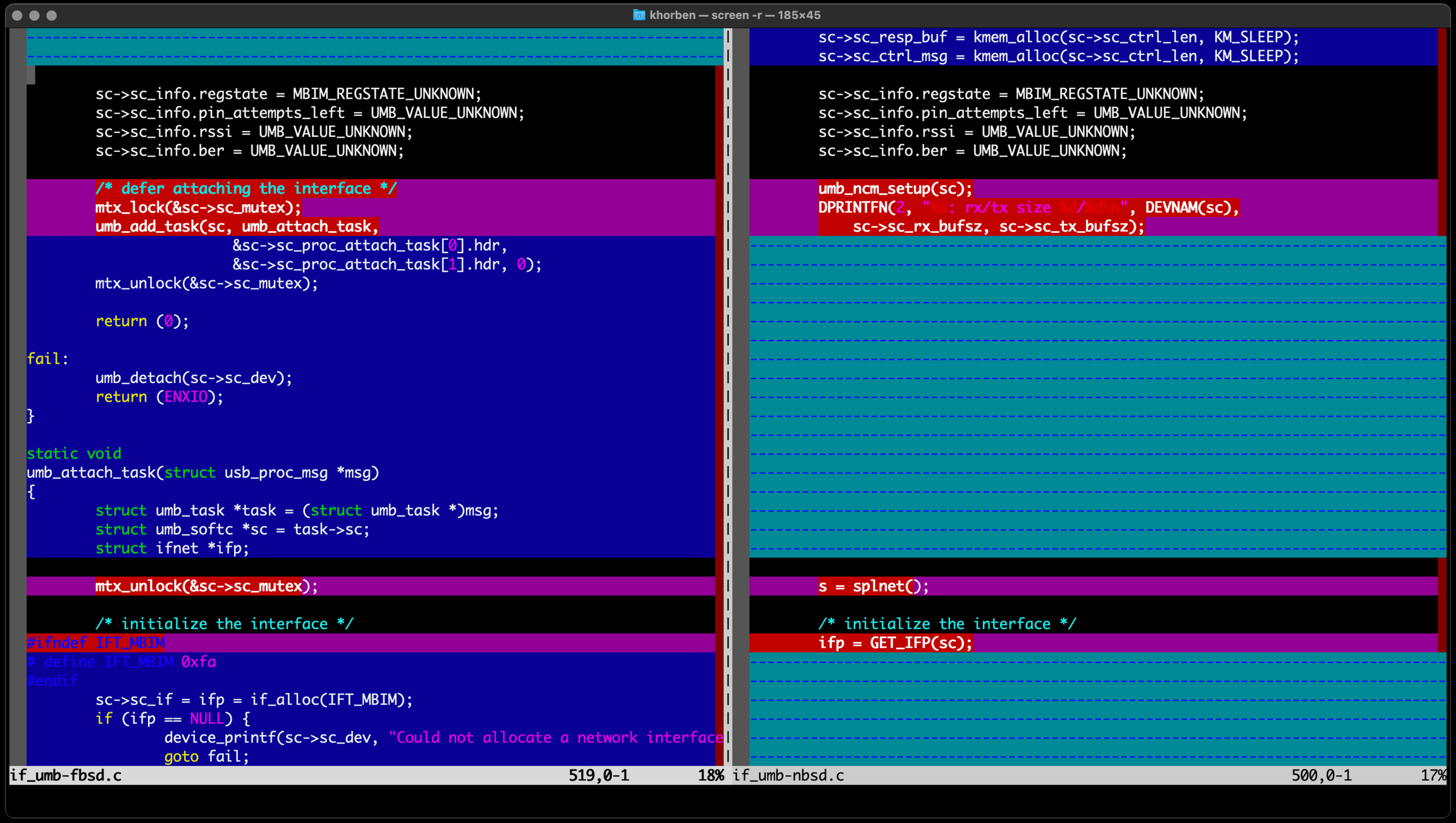

khorben — screen -r — 185×45
/*
* Some MBIM hardware does not provide the mandatory CDC Union
* Descriptor, so we also look at matching Interface
* Association Descriptors to find out the MBIM Data Interface
* number.
*/
sc->sc_ver_maj = sc->sc_ver_min = -1;
sc->sc_maxpktlen = MBIM_MAXSEGSZ_MINVAL;
id = usbd_get_interface_descriptor(uaa->iface);
ud = usbd_find_descriptor(sc->sc_udev, id, uaa->info.bIfaceIndex,
UDESC_CS_INTERFACE, 0xff, UDESCSUB_CDC_UNION, 0xff);
if (ud != NULL) {
data_ifaceno = ud->bSlaveInterface[0];
}
md = usbd_find_descriptor(sc->sc_udev, id, uaa->info.bIfaceIndex,
UDESC_CS_INTERFACE, 0xff, UDESCSUB_MBIM, 0xff);
if (md != NULL) {
v = UGETW(md->bcdMBIMVersion);
sc->sc_ver_maj = MBIM_VER_MAJOR(v);
sc->sc_ver_min = MBIM_VER_MINOR(v);
sc->sc_ctrl_len = UGETW(md->wMaxControlMessage);
/* Never trust a USB device! Could try to exploit us */
if (sc->sc_ctrl_len < MBIM_CTRLMSG_MINLEN ||
sc->sc_ctrl_len > MBIM_CTRLMSG_MAXLEN) {
DPRINTF("control message len %d out of "
"bounds [%d .. %d]\n",
sc->sc_ctrl_len, MBIM_CTRLMSG_MINLEN,
MBIM_CTRLMSG_MAXLEN);
/* continue anyway */
}
sc->sc_maxpktlen = UGETW(md->wMaxSegmentSize);
DPRINTFN(2, "ctrl_len=%d, maxpktlen=%d, cap=0x%x\n",
sc->sc_ctrl_len, sc->sc_maxpktlen,
md->bmNetworkCapabilities);
if_umb-fbsd.c 398,2-9 13%
/*
* Some MBIM hardware does not provide the mandatory CDC Union
* Descriptor, so we also look at matching Interface
* Association Descriptors to find out the MBIM Data Interface
* number.
*/
sc->sc_ver_maj = sc->sc_ver_min = -1;
sc->sc_maxpktlen = MBIM_MAXSEGSZ_MINVAL;
usb_desc_iter_init(sc->sc_udev, &iter);
while ((desc = usb_desc_iter_next(&iter))) {
if (desc->bDescriptorType == UDESC_INTERFACE_ASSOC) {
ad = (const usb_interface_assoc_descriptor_t *)desc;
if (ad->bFirstInterface == uiaa->uiaa_ifaceno &&
ad->bInterfaceCount > 1)
data_ifaceno = uiaa->uiaa_ifaceno + 1;
continue;
}
if (desc->bDescriptorType == UDESC_INTERFACE) {
id = (const usb_interface_descriptor_t *)desc;
current_ifaceno = id->bInterfaceNumber;
continue;
}
if (current_ifaceno != uiaa->uiaa_ifaceno)
continue;
if (desc->bDescriptorType != UDESC_CS_INTERFACE)
continue;
switch (desc->bDescriptorSubtype) {
case UDESCSUB_CDC_UNION:
ud = (const usb_cdc_union_descriptor_t *)desc;
data_ifaceno = ud->bSlaveInterface[0];
break;
case UDESCSUB_MBIM:
md = (const struct mbim_descriptor *)desc;
v = UGETW(md->bcdMBIMVersion);
sc->sc_ver_maj = MBIM_VER_MAJOR(v);
sc->sc_ver_min = MBIM_VER_MINOR(v);
sc->sc_ctrl_len = UGETW(md->wMaxControlMessage);
/* Never trust a USB device! Could try to exploit us */
if (sc->sc_ctrl_len < MBIM_CTRLMSG_MINLEN ||
if_umb-nbsd.c 328,2-9 11%

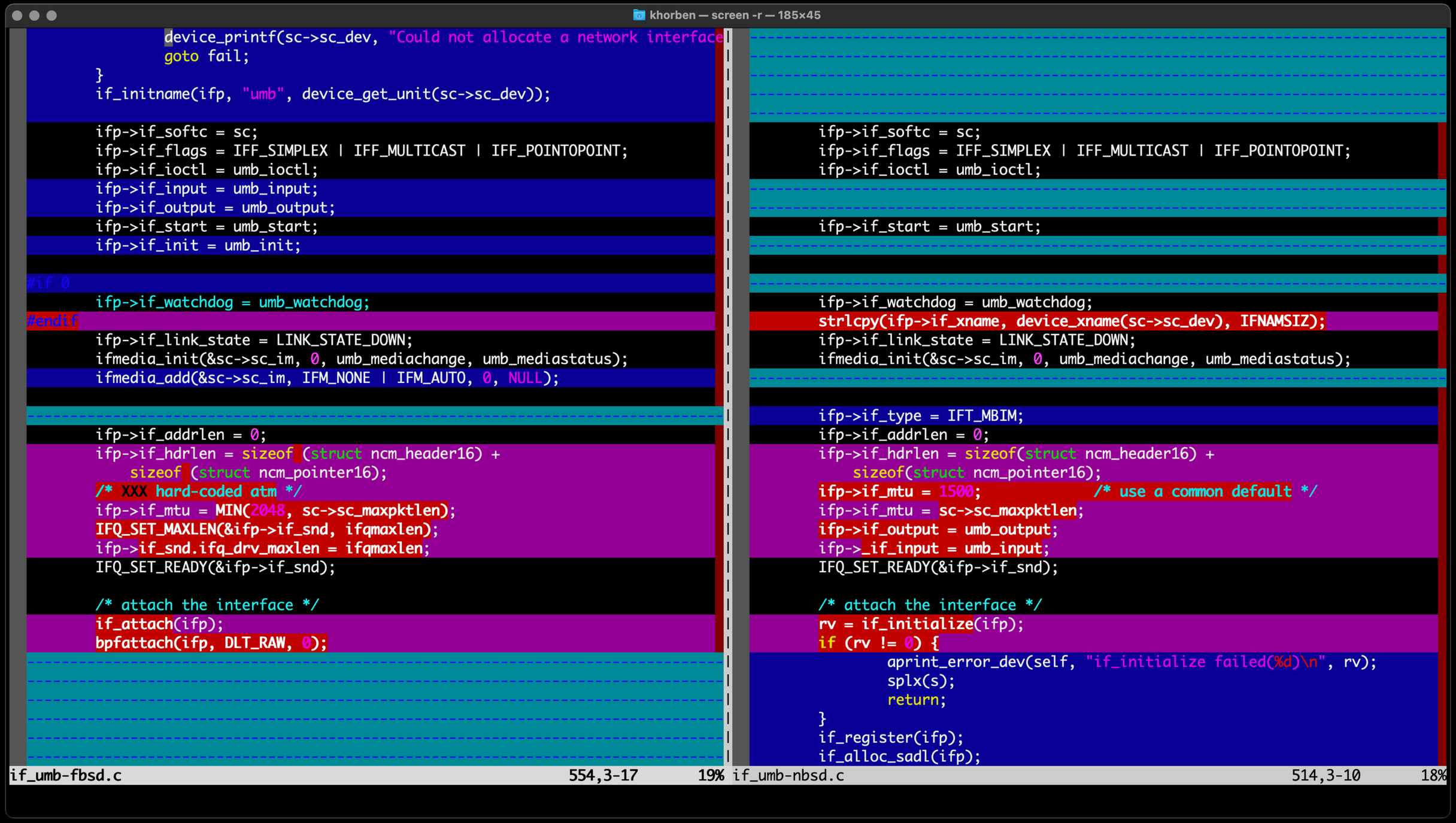

khorben — screen -r — 185×45
}
if (sc->sc_ver_maj < 0) {
device_printf(dev, "error: missing MBIM descriptor\n");
goto fail;
}
device_printf(dev, "version %d.%d\n", sc->sc_ver_maj,
sc->sc_ver_min);
if (usbd_lookup_id_by_uaa(umb_fccauth_devs, sizeof (umb_fccauth_devs), ua
sc->sc_flags |= UMBFLG_FCC_AUTH_REQUIRED;
sc->sc_cid = -1;
}
for (i = 0; i < sc->sc_udev->ifaces_max; i++) {
iface = usbd_get_iface(sc->sc_udev, i);
id = usbd_get_interface_descriptor(iface);
if (id == NULL)
break;
if (id->bInterfaceNumber == data_ifaceno) {
sc->sc_data_iface = iface;
if_umb-fbsd.c 434,2-9 15%
/* Never trust a USB device! Could try to exploit us */
if (sc->sc_ctrl_len < MBIM_CTRLMSG_MINLEN ||
sc->sc_ctrl_len > MBIM_CTRLMSG_MAXLEN) {
DPRINTF("%s: control message len %d out of "
"bounds [%d .. %d]\n", DEVNAM(sc),
sc->sc_ctrl_len, MBIM_CTRLMSG_MINLEN,
MBIM_CTRLMSG_MAXLEN);
/* cont. anyway */
}
sc->sc_maxpktlen = UGETW(md->wMaxSegmentSize);
DPRINTFN(2, "%s: ctrl_len=%d, maxpktlen=%d, cap=0x%x\n",
DEVNAM(sc), sc->sc_ctrl_len, sc->sc_maxpktlen,
md->bmNetworkCapabilities);
break;
default:
break;
}
}
if (sc->sc_ver_maj < 0) {
aprint_error_dev(self, "missing MBIM descriptor\n");
goto fail;
}
aprint_normal_dev(self, "version %d.%d\n", sc->sc_ver_maj,
sc->sc_ver_min);
if (usb_lookup(umb_fccauth_devs, uiaa->uiaa_vendor, uiaa->uiaa_product))
sc->sc_flags |= UMBFLG_FCC_AUTH_REQUIRED;
sc->sc_cid = -1;
}
for (i = 0; i < uiaa->uiaa_nifaces; i++) {
id = usbd_get_interface_descriptor(uiaa->uiaa_ifaces[i]);
if (id != NULL && id->bInterfaceNumber == data_ifaceno) {
sc->sc_data_iface = uiaa->uiaa_ifaces[i];
if_umb-nbsd.c 382,2-9 12%

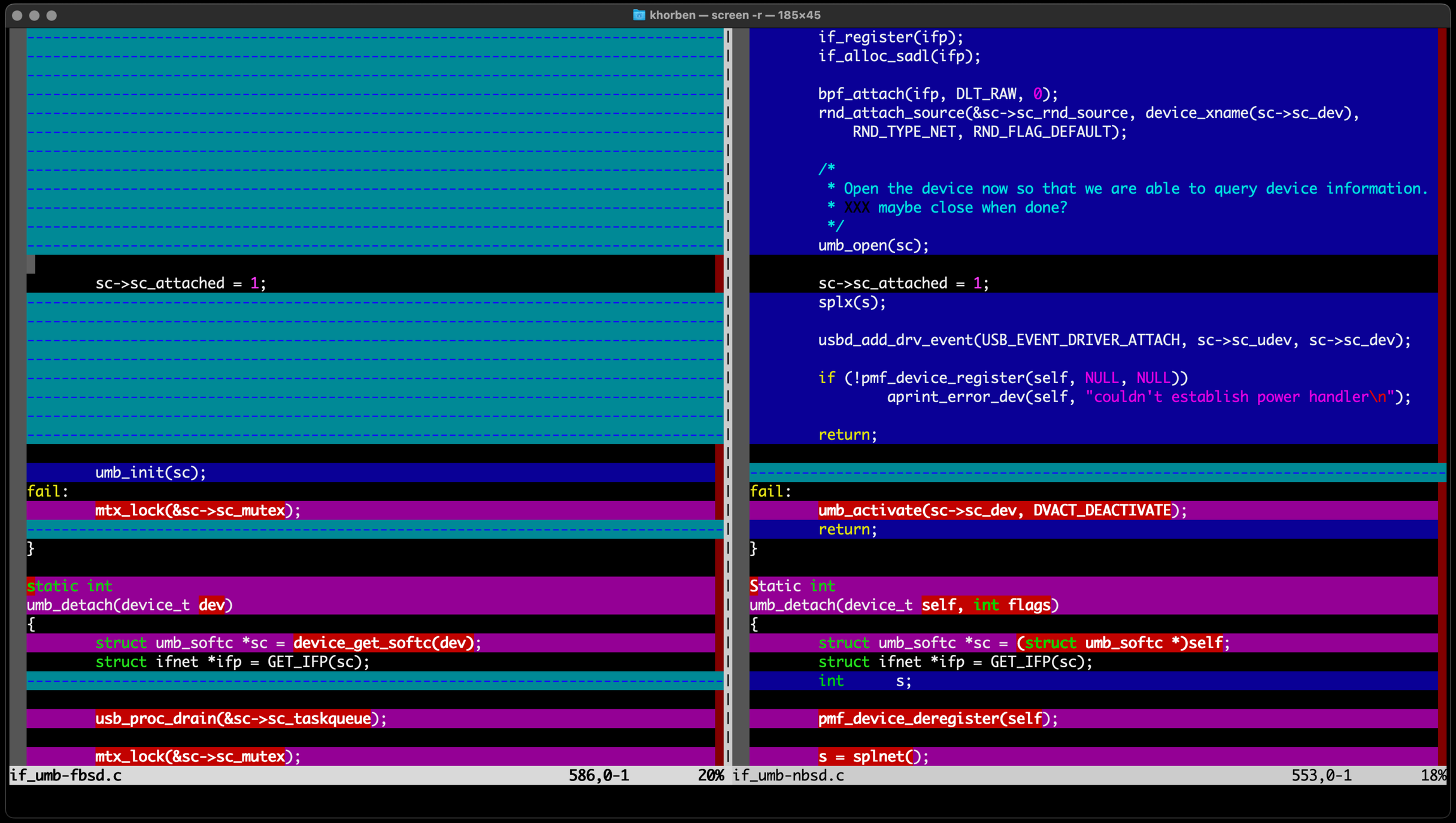

khorben — screen -r — 185×45
if (id->bInterfaceNumber == data_ifaceno) {
sc->sc_data_iface = iface;
sc->sc_ifaces_index[0] = i;
sc->sc_ifaces_index[1] = uaa->info.bIfaceIndex;
break;
}
}
if (sc->sc_data_iface == NULL) {
device_printf(dev, "error: no data interface found\n");
goto fail;
}
/*
* If this is a combined NCM/MBIM function, switch to
* alternate setting one to enable MBIM.
*/
id = usbd_get_interface_descriptor(uaa->iface);
if (id != NULL && id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass == UISUBCLASS_NETWORK_CONTROL_MODEL) {
device_printf(sc->sc_dev, "combined NCM/MBIM\n");
error = usbd_req_set_alt_interface_no(sc->sc_udev,
NULL, uaa->info.bIfaceIndex, 1);
if (error != USB_ERR_NORMAL_COMPLETION) {
device_printf(dev, "error: Could not switch to alternate
goto fail;
}
sc->sc_ifaces_index[1] = uaa->info.bIfaceIndex + 1;
}
if (usb_proc_create(&sc->sc_taskqueue, &sc->sc_mutex,
device_get_nameunit(sc->sc_dev),
USB_PRI_MED) != 0)
if_umb-fbsd.c 454,3-17 15%
}
}
if (sc->sc_data_iface == NULL) {
aprint_error_dev(self, "no data interface found\n");
goto fail;
}
/*
* If this is a combined NCM/MBIM function, switch to
* alternate setting one to enable MBIM.
*/
id = usbd_get_interface_descriptor(uiaa->uiaa_iface);
if (id->bInterfaceClass == UICLASS_CDC &&
id->bInterfaceSubClass ==
UISUBCLASS_NETWORK_CONTROL_MODEL)
usbd_set_interface(uiaa->uiaa_iface, 1);
id = usbd_get_interface_descriptor(uiaa->uiaa_iface);
ctrl_ep = -1;
for (i = 0; i < id->bNumEndpoints && ctrl_ep == -1; i++) {
ed = usbd_interface2endpoint_descriptor(uiaa->uiaa_iface, i);
if (ed == NULL)
break;
if (UE_GET_XFERTYPE(ed->bmAttributes) == UE_INTERRUPT &&
UE_GET_DIR(ed->bEndpointAddress) == UE_DIR_IN)
ctrl_ep = ed->bEndpointAddress;
}
if_umb-nbsd.c 400,3-17 14%

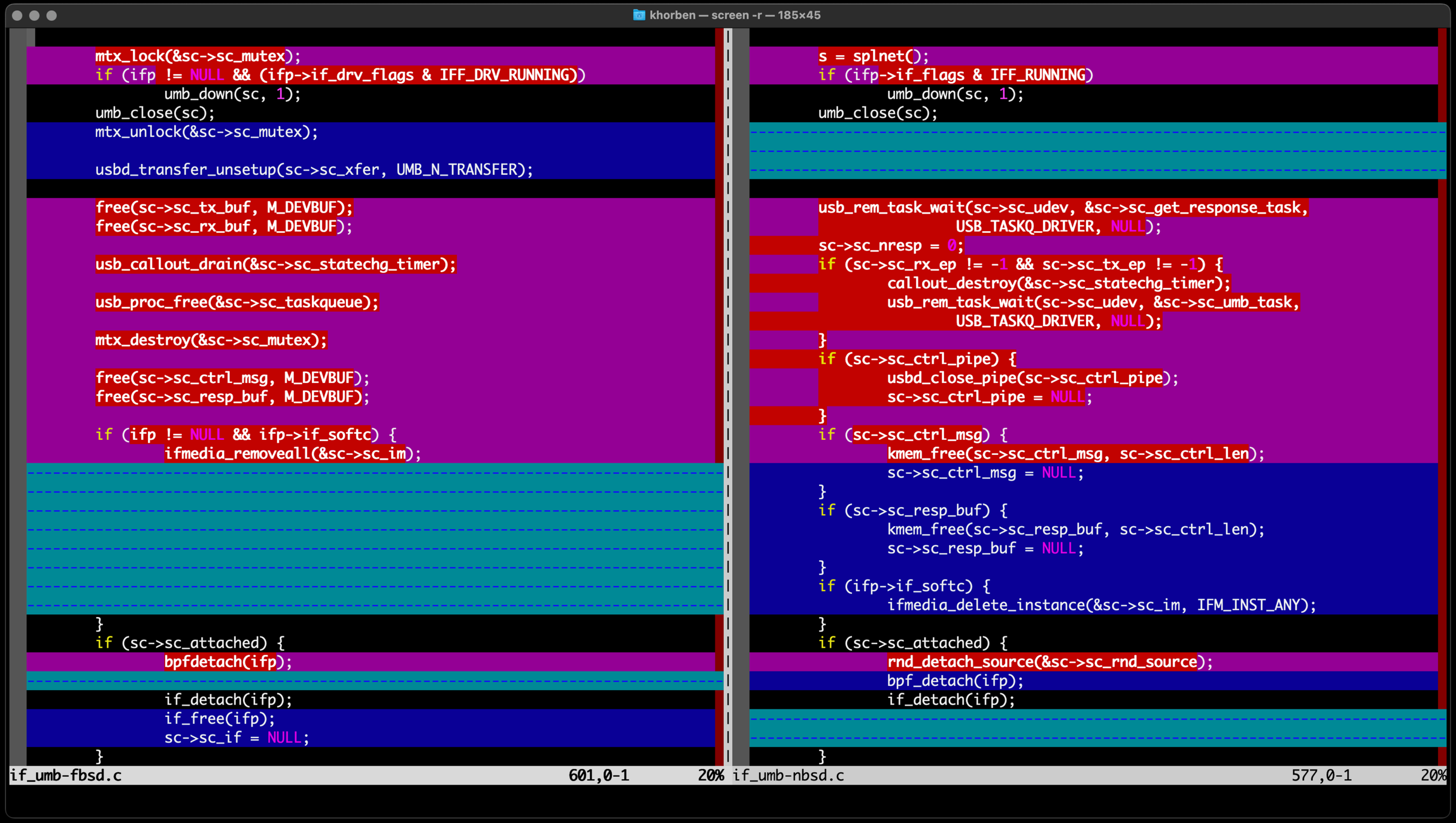

khorben — screen -r — 185×45
goto fail;
DPRINTFN(2, "ctrl-ifno#%d: data-ifno#%d\n", sc->sc_ctrl_ifaceno,
data_ifaceno);
usb_callout_init_mtx(&sc->sc_statechg_timer, &sc->sc_mutex, 0);
umb_ncm_setup(sc, config);
DPRINTFN(2, "%s: rx/tx size %d/%d\n", DEVNAM(sc),
sc->sc_rx_bufsz, sc->sc_tx_bufsz);
sc->sc_rx_buf = malloc(sc->sc_rx_bufsz, M_DEVBUF, M_WAITOK);
sc->sc_tx_buf = malloc(sc->sc_tx_bufsz, M_DEVBUF, M_WAITOK);
for (i = 0; i != 32; i++) {
error = usbd_set_alt_interface_index(sc->sc_udev,
sc->sc_ifaces_index[0], i);
if (error)
break;
error = usbd_transfer_setup(sc->sc_udev, sc->sc_ifaces_index,
sc->sc_xfer, config, UMB_N_TRANSFER,
if_umb-fbsd.c 486,3-17 17%
ctrl_ep = ed->bEndpointAddress;
}
if (ctrl_ep == -1) {
aprint_error_dev(self, "missing interrupt endpoint\n");
goto fail;
}
/*
* For the MBIM Data Interface, select the appropriate
* alternate setting by looking for a matching descriptor that
* has two endpoints.
*/
cd = usbd_get_config_descriptor(sc->sc_udev);
altnum = usbd_get_no_alts(cd, data_ifaceno);
for (i = 0; i < altnum; i++) {
id = usbd_find_idesc(cd, sc->sc_data_iface->ui_index, i);
if (id == NULL)
continue;
if (id->bInterfaceClass == UICLASS_CDC_DATA &&
id->bInterfaceSubClass == UISUBCLASS_DATA &&
id->bInterfaceProtocol == UIPROTO_DATA_MBIM &&
id->bNumEndpoints == 2)
break;
}
if (i == altnum || id == NULL) {
aprint_error_dev(self, "missing alt setting for interface #%d\n",
data_ifaceno);
goto fail;
}
status = usbd_set_interface(sc->sc_data_iface, i);
if (status) {
aprint_error_dev(self, "select alt setting %d for interface #%d "
"failed: %s\n", i, data_ifaceno, usbd_errstr(status));
goto fail;
}
id = usbd_get_interface_descriptor(sc->sc_data_iface);
sc->sc_rx_ep = sc->sc_tx_ep = -1;
if_umb-nbsd.c 429,3-17 14%

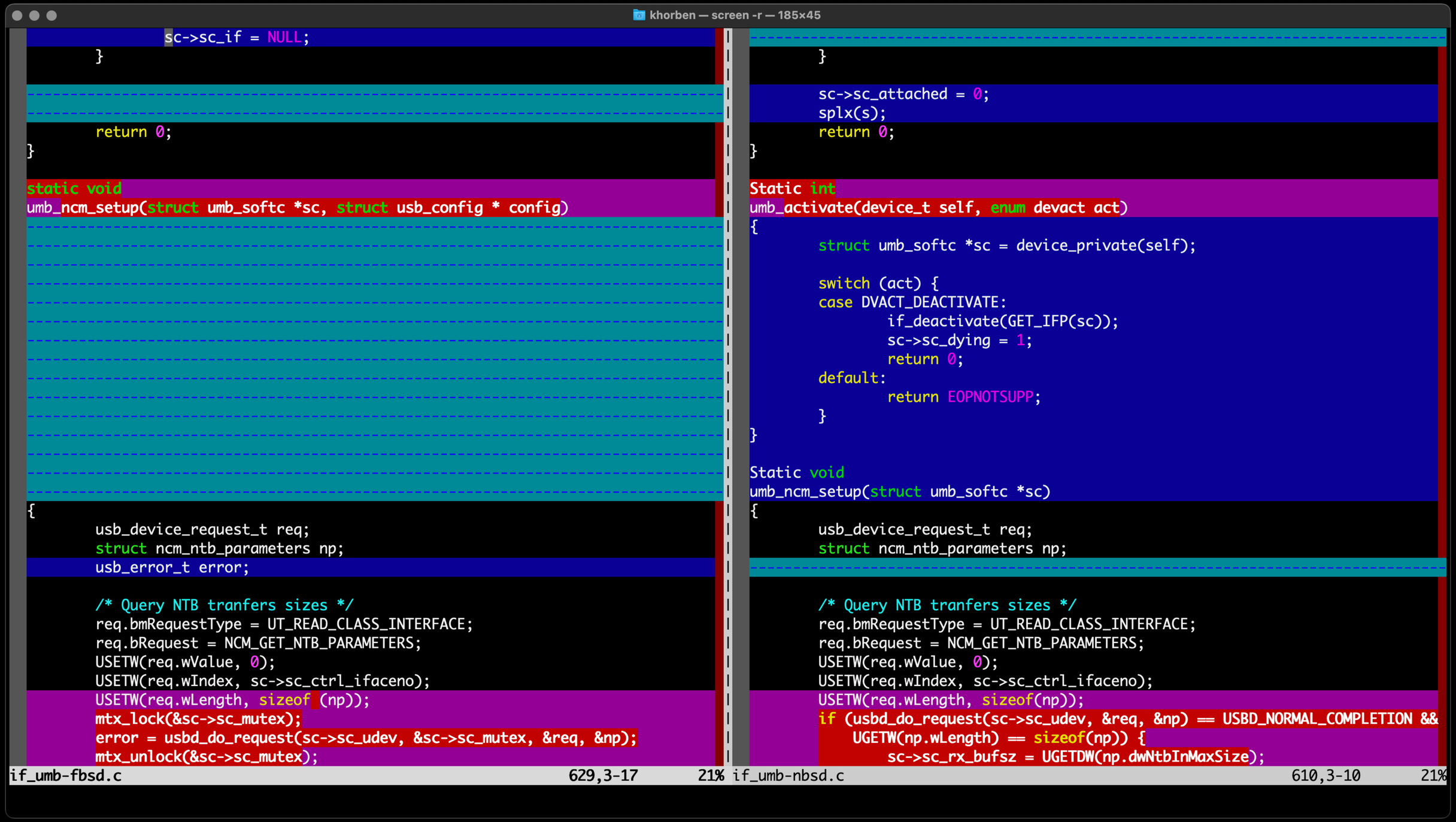

khorben — screen -r — 185×45
error = usbd_transfer_setup(sc->sc_udev, sc->sc_ifaces_index,
sc->sc_xfer, config, UMB_N_TRANSFER,
sc, &sc->sc_mutex);
if (error == USB_ERR_NORMAL_COMPLETION)
break;
}
if (error || (i == 32)) {
device_printf(sc->sc_dev, "error: failed to setup xfers\n");
goto fail;
}
sc->sc_resp_buf = malloc(sc->sc_ctrl_len, M_DEVBUF, M_WAITOK);
sc->sc_ctrl_msg = malloc(sc->sc_ctrl_len, M_DEVBUF, M_WAITOK);
if_umb-fbsd.c 506,3-17 17%
id = usbd_get_interface_descriptor(sc->sc_data_iface);
sc->sc_rx_ep = sc->sc_tx_ep = -1;
for (i = 0; i < id->bNumEndpoints; i++) {
if ((ed = usbd_interface2endpoint_descriptor(sc->sc_data_iface,
i)) == NULL)
break;
if (UE_GET_XFERTYPE(ed->bmAttributes) == UE_BULK &&
UE_GET_DIR(ed->bEndpointAddress) == UE_DIR_IN)
sc->sc_rx_ep = ed->bEndpointAddress;
else if (UE_GET_XFERTYPE(ed->bmAttributes) == UE_BULK &&
UE_GET_DIR(ed->bEndpointAddress) == UE_DIR_OUT)
sc->sc_tx_ep = ed->bEndpointAddress;
}
if (sc->sc_rx_ep == -1 || sc->sc_tx_ep == -1) {
aprint_error_dev(self, "missing bulk endpoints\n");
goto fail;
}
DPRINTFN(2, "%s: ctrl-ifno#%d: ep-ctrl=%d, data-ifno#%d: ep-rx=%d, "
"ep-tx=%d\n", DEVNAM(sc), sc->sc_ctrl_ifaceno,
UE_GET_ADDR(ctrl_ep), data_ifaceno,
UE_GET_ADDR(sc->sc_rx_ep), UE_GET_ADDR(sc->sc_tx_ep));
usb_init_task(&sc->sc_umb_task, umb_state_task, sc,
0);
usb_init_task(&sc->sc_get_response_task, umb_get_response_task, sc,
0);
callout_init(&sc->sc_statechg_timer, 0);
callout_setfunc(&sc->sc_statechg_timer, umb_statechg_timeout, sc);
if (usbd_open_pipe_intr(uiaa->uiaa_iface, ctrl_ep, USBD_SHORT_XFER_OK,
&sc->sc_ctrl_pipe, sc, &sc->sc_intr_msg, sizeof(sc->sc_intr_msg),
umb_intr, USBD_DEFAULT_INTERVAL)) {
aprint_error_dev(self, "failed to open control pipe\n");
goto fail;
}
sc->sc_resp_buf = kmem_alloc(sc->sc_ctrl_len, KM_SLEEP);
sc->sc_ctrl_msg = kmem_alloc(sc->sc_ctrl_len, KM_SLEEP);
if_umb-nbsd.c 461,3-10 16%

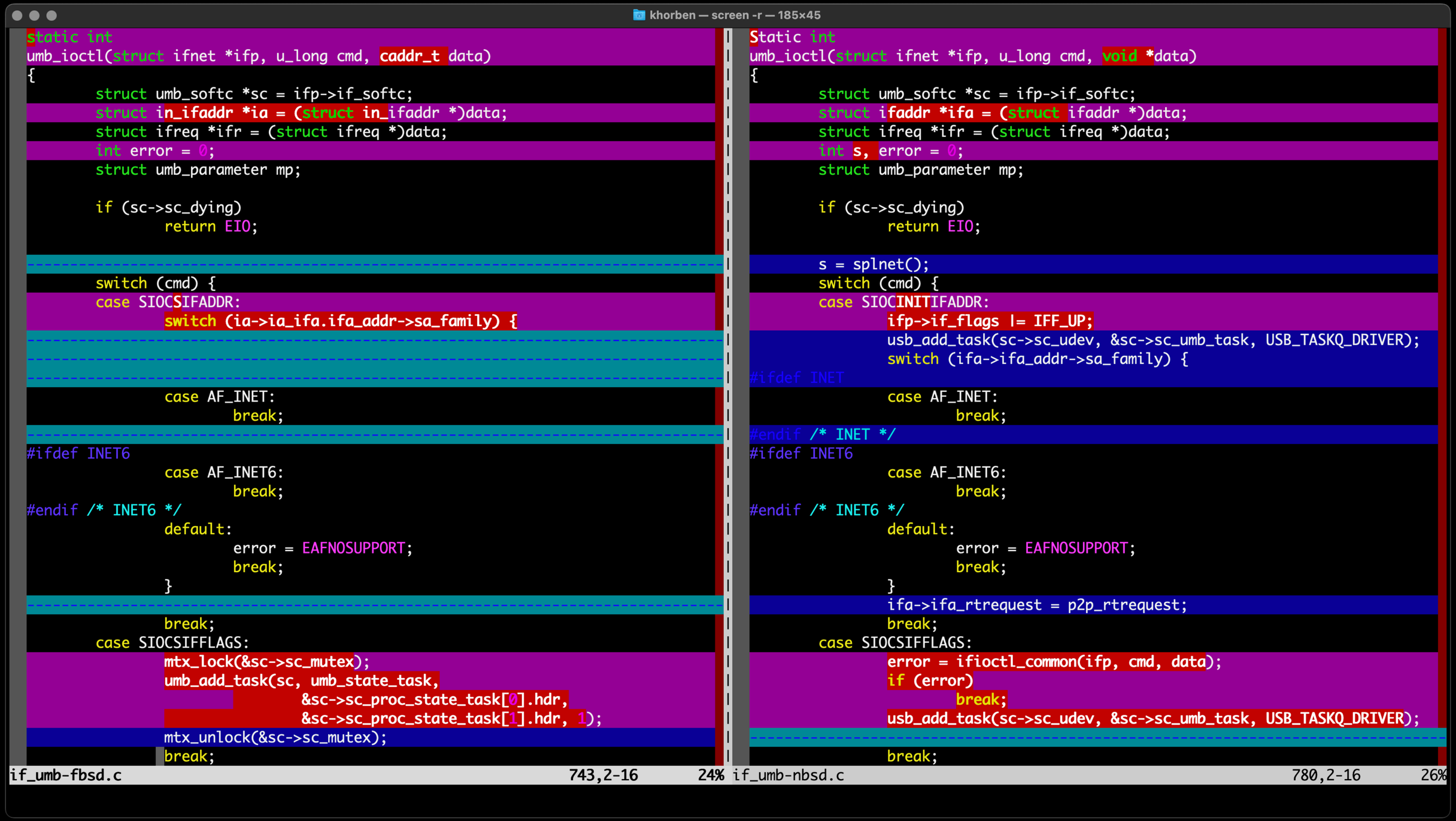

khorben — screen -r — 185×45
sc->sc_info.regstate = MBIM_REGSTATE_UNKNOWN;
sc->sc_info.pin_attempts_left = UMB_VALUE_UNKNOWN;
sc->sc_info.rssi = UMB_VALUE_UNKNOWN;
sc->sc_info.ber = UMB_VALUE_UNKNOWN;
/* defer attaching the interface */
mtx_lock(&sc->sc_mutex);
umb_add_task(sc, umb_attach_task,
&sc->sc_proc_attach_task[0].hdr,
&sc->sc_proc_attach_task[1].hdr, 0);
mtx_unlock(&sc->sc_mutex);
return (0);
fail:
umb_detach(sc->sc_dev);
return (ENXIO);
}
static void
umb_attach_task(struct usb_proc_msg *msg)
{
struct umb_task *task = (struct umb_task *)msg;
struct umb_softc *sc = task->sc;
struct ifnet *ifp;
mtx_unlock(&sc->sc_mutex);
/* initialize the interface */
#ifndef IFT_MBIM
define IFT_MBIM 0xfa
#endif
sc->sc_if = ifp = if_alloc(IFT_MBIM);
if (ifp == NULL) {
device_printf(sc->sc_dev, "Could not allocate a network interface
goto fail;
if_umb-fbsd.c 519,0-1 18%
sc->sc_resp_buf = kmem_alloc(sc->sc_ctrl_len, KM_SLEEP);
sc->sc_ctrl_msg = kmem_alloc(sc->sc_ctrl_len, KM_SLEEP);
sc->sc_info.regstate = MBIM_REGSTATE_UNKNOWN;
sc->sc_info.pin_attempts_left = UMB_VALUE_UNKNOWN;
sc->sc_info.rssi = UMB_VALUE_UNKNOWN;
sc->sc_info.ber = UMB_VALUE_UNKNOWN;
umb_ncm_setup(sc);
DPRINTFN(2, "%s: rx/tx size %d/%d\n", DEVNAM(sc),
sc->sc_rx_bufsz, sc->sc_tx_bufsz);
s = splnet();
/* initialize the interface */
ifp = GET_IFP(sc);
if_umb-nbsd.c 500,0-1 17%

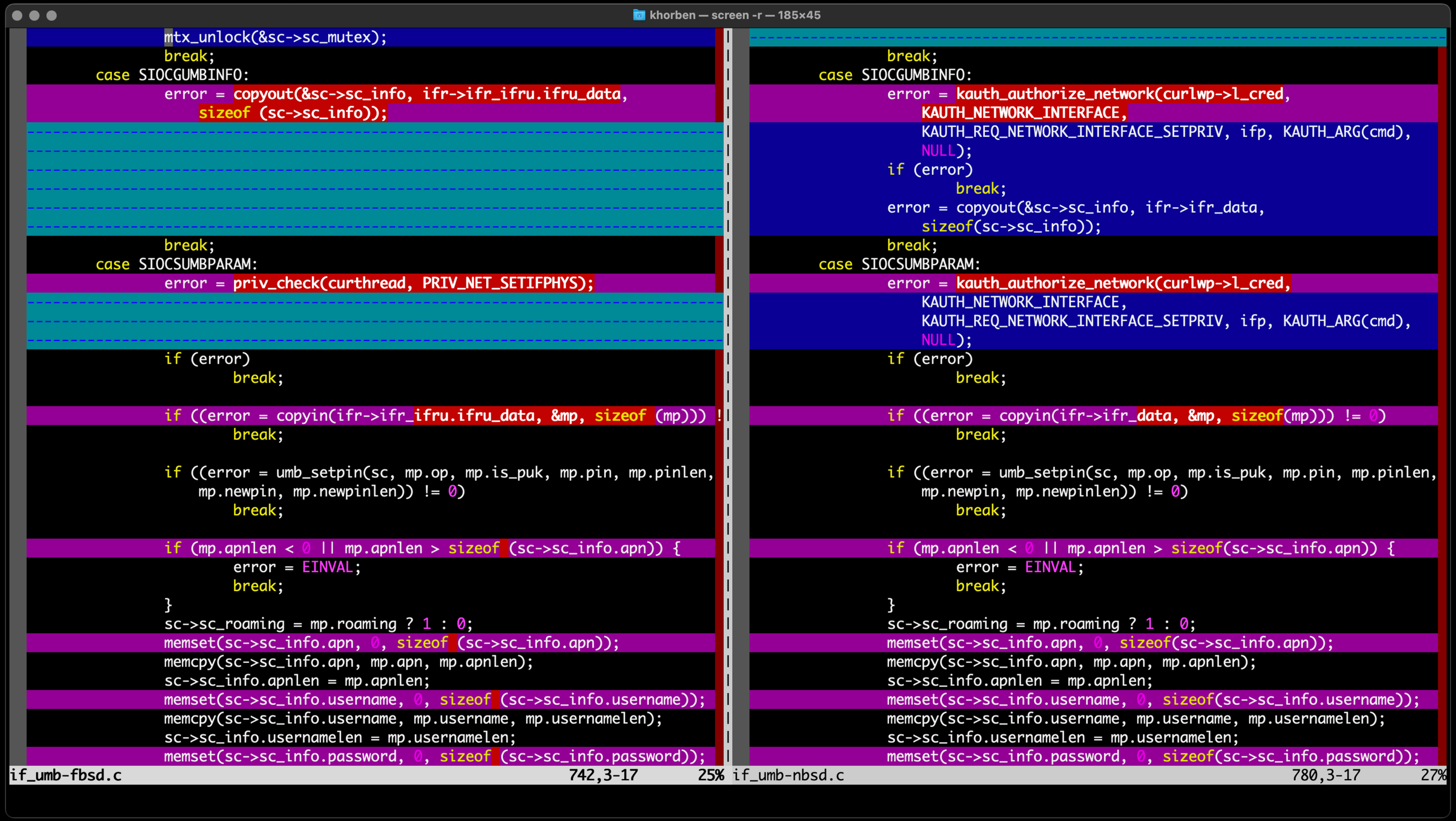

khorben — screen -r — 185×45
device_printf(sc->sc_dev, "Could not allocate a network interface
goto fail;
}
if_initname(ifp, "umb", device_get_unit(sc->sc_dev));
ifp->if_softc = sc;
ifp->if_flags = IFF_SIMPLEX | IFF_MULTICAST | IFF_POINTOPOINT;
ifp->if_ioctl = umb_ioctl;
ifp->if_input = umb_input;
ifp->if_output = umb_output;
ifp->if_start = umb_start;
ifp->if_init = umb_init;
#if 0
ifp->if_watchdog = umb_watchdog;
#endif
ifp->if_link_state = LINK_STATE_DOWN;
ifmedia_init(&sc->sc_im, 0, umb_mediachange, umb_mediastatus);
ifmedia_add(&sc->sc_im, IFM_NONE | IFM_AUTO, 0, NULL);
ifp->if_addrlen = 0;
ifp->if_hdrlen = sizeof (struct ncm_header16) +
sizeof (struct ncm_pointer16);
/* XXX hard-coded atm */
ifp->if_mtu = MIN(2048, sc->sc_maxpktlen);
IFQ_SET_MAXLEN(&ifp->if_snd, ifqmaxlen);
ifp->if_snd.ifq_drv_maxlen = ifqmaxlen;
IFQ_SET_READY(&ifp->if_snd);
/* attach the interface */
if_attach(ifp);
bpfattach(ifp, DLT_RAW, 0);
if_umb-fbsd.c 554,3-17 19%
ifp->if_softc = sc;
ifp->if_flags = IFF_SIMPLEX | IFF_MULTICAST | IFF_POINTOPOINT;
ifp->if_ioctl = umb_ioctl;
ifp->if_start = umb_start;
ifp->if_watchdog = umb_watchdog;
strlcpy(ifp->if_xname, device_xname(sc->sc_dev), IFNAMSIZ);
ifp->if_link_state = LINK_STATE_DOWN;
ifmedia_init(&sc->sc_im, 0, umb_mediachange, umb_mediastatus);
ifp->if_type = IFT_MBIM;
ifp->if_addrlen = 0;
ifp->if_hdrlen = sizeof(struct ncm_header16) +
sizeof(struct ncm_pointer16);
ifp->if_mtu = 1500; /* use a common default */
ifp->if_mtu = sc->sc_maxpktlen;
ifp->if_output = umb_output;
ifp->_if_input = umb_input;
IFQ_SET_READY(&ifp->if_snd);
/* attach the interface */
rv = if_initialize(ifp);
if (rv != 0) {
aprint_error_dev(self, "if_initialize failed(%d)\n", rv);
splx(s);
return;
}
if_register(ifp);
if_alloc_sadl(ifp);
if_umb-nbsd.c 514,3-10 18%

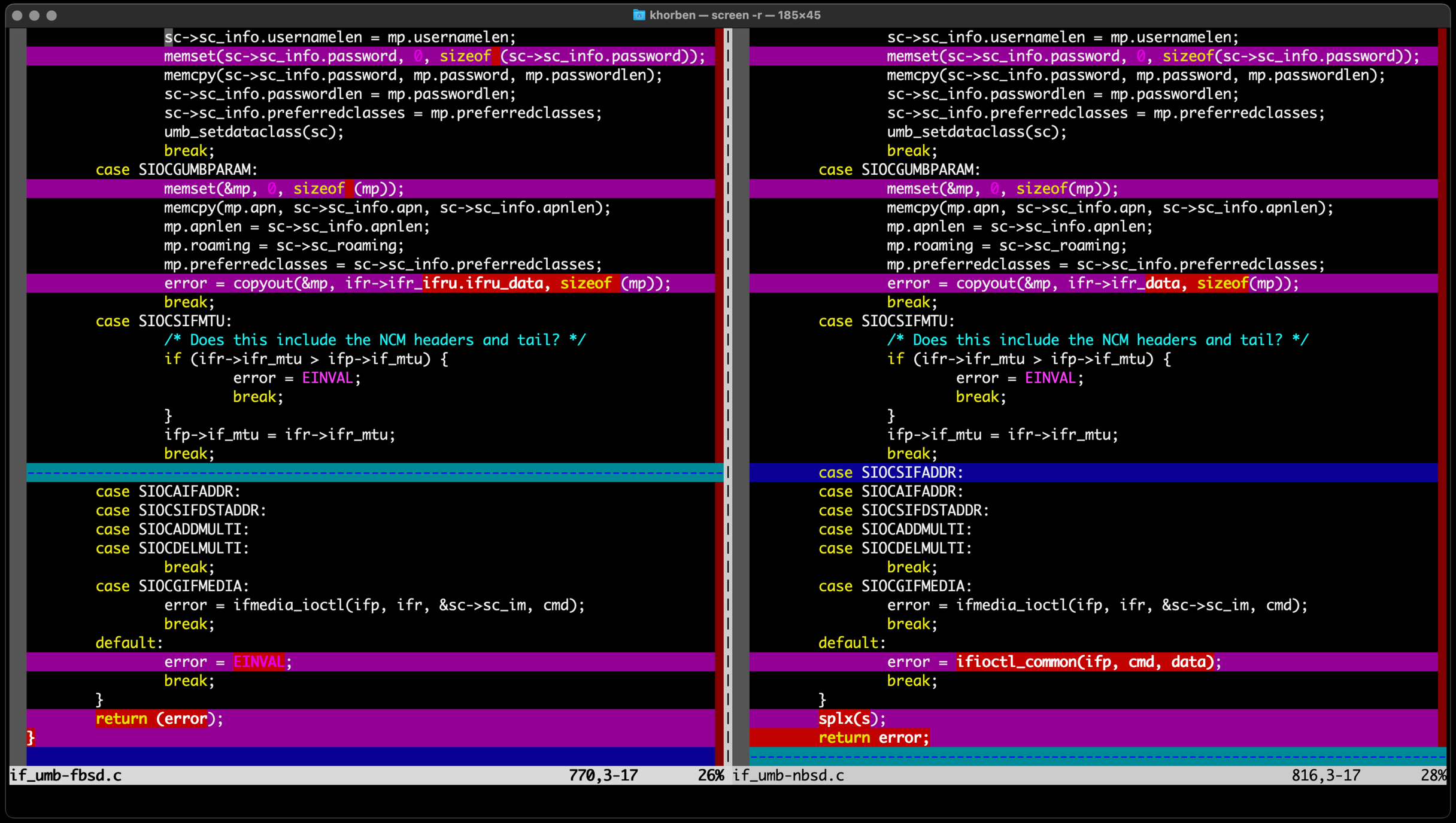

khorben — screen -r — 185×45
sc->sc_attached = 1;
umb_init(sc);
fail:
mtx_lock(&sc->sc_mutex);
}
static int
umb_detach(device_t dev)
{
struct umb_softc *sc = device_get_softc(dev);
struct ifnet *ifp = GET_IFP(sc);
usb_proc_drain(&sc->sc_taskqueue);
mtx_lock(&sc->sc_mutex);
if_umb-fbsd.c 586,0-1 20%
if_register(ifp);
if_alloc_sadl(ifp);
bpf_attach(ifp, DLT_RAW, 0);
rnd_attach_source(&sc->sc_rnd_source, device_xname(sc->sc_dev),
RND_TYPE_NET, RND_FLAG_DEFAULT);
/*
* Open the device now so that we are able to query device information.
* XXX maybe close when done?
*/
umb_open(sc);
sc->sc_attached = 1;
splx(s);
usbd_add_drv_event(USB_EVENT_DRIVER_ATTACH, sc->sc_udev, sc->sc_dev);
if (!pmf_device_register(self, NULL, NULL))
aprint_error_dev(self, "couldn't establish power handler\n");
return;
fail:
umb_activate(sc->sc_dev, DVACT_DEACTIVATE);
return;
}
Static int
umb_detach(device_t self, int flags)
{
struct umb_softc *sc = (struct umb_softc *)self;
struct ifnet *ifp = GET_IFP(sc);
int s;
pmf_device_deregister(self);
s = splnet();
if_umb-nbsd.c 553,0-1 18%

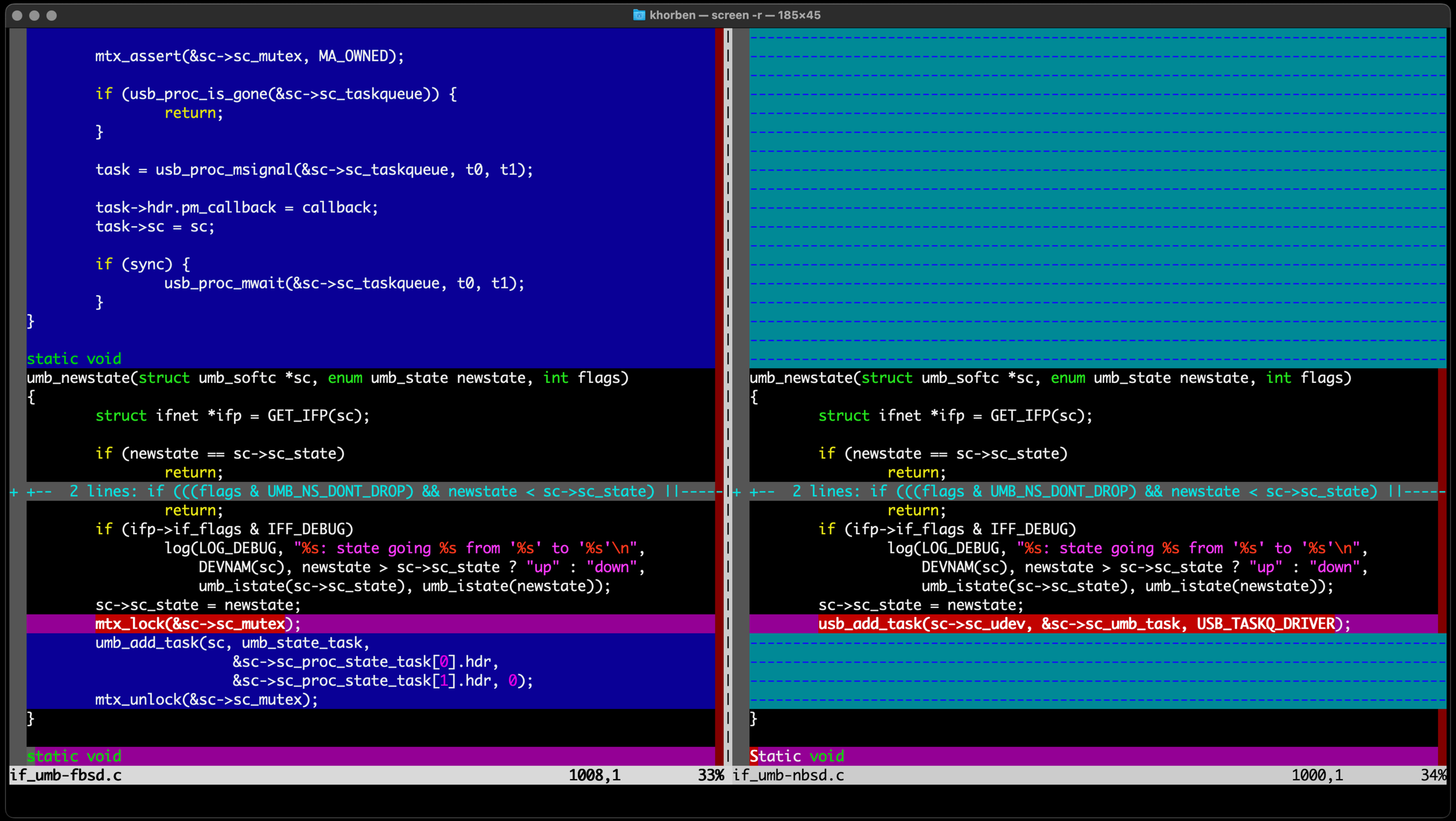

khorben — screen -r — 185×45
mtx_lock(&sc->sc_mutex);
if (ifp != NULL && (ifp->if_drv_flags & IFF_DRV_RUNNING))
umb_down(sc, 1);
umb_close(sc);
mtx_unlock(&sc->sc_mutex);
usbd_transfer_unsetup(sc->sc_xfer, UMB_N_TRANSFER);
free(sc->sc_tx_buf, M_DEVBUF);
free(sc->sc_rx_buf, M_DEVBUF);
usb_callout_drain(&sc->sc_statechg_timer);
usb_proc_free(&sc->sc_taskqueue);
mtx_destroy(&sc->sc_mutex);
free(sc->sc_ctrl_msg, M_DEVBUF);
free(sc->sc_resp_buf, M_DEVBUF);
if (ifp != NULL && ifp->if_softc) {
ifmedia_removeall(&sc->sc_im);
}
if (sc->sc_attached) {
bpfdetach(ifp);
if_detach(ifp);
if_free(ifp);
sc->sc_if = NULL;
}
if_umb-fbsd.c 601,0-1 20%
s = splnet();
if (ifp->if_flags & IFF_RUNNING)
umb_down(sc, 1);
umb_close(sc);
usb_rem_task_wait(sc->sc_udev, &sc->sc_get_response_task,
USB_TASKQ_DRIVER, NULL);
sc->sc_nresp = 0;
if (sc->sc_rx_ep != -1 && sc->sc_tx_ep != -1) {
callout_destroy(&sc->sc_statechg_timer);
usb_rem_task_wait(sc->sc_udev, &sc->sc_umb_task,
USB_TASKQ_DRIVER, NULL);
}
if (sc->sc_ctrl_pipe) {
usbd_close_pipe(sc->sc_ctrl_pipe);
sc->sc_ctrl_pipe = NULL;
}
if (sc->sc_ctrl_msg) {
kmem_free(sc->sc_ctrl_msg, sc->sc_ctrl_len);
sc->sc_ctrl_msg = NULL;
}
if (sc->sc_resp_buf) {
kmem_free(sc->sc_resp_buf, sc->sc_ctrl_len);
sc->sc_resp_buf = NULL;
}
if (ifp->if_softc) {
ifmedia_delete_instance(&sc->sc_im, IFM_INST_ANY);
}
if (sc->sc_attached) {
rnd_detach_source(&sc->sc_rnd_source);
bpf_detach(ifp);
if_detach(ifp);
}
if_umb-nbsd.c 577,0-1 20%

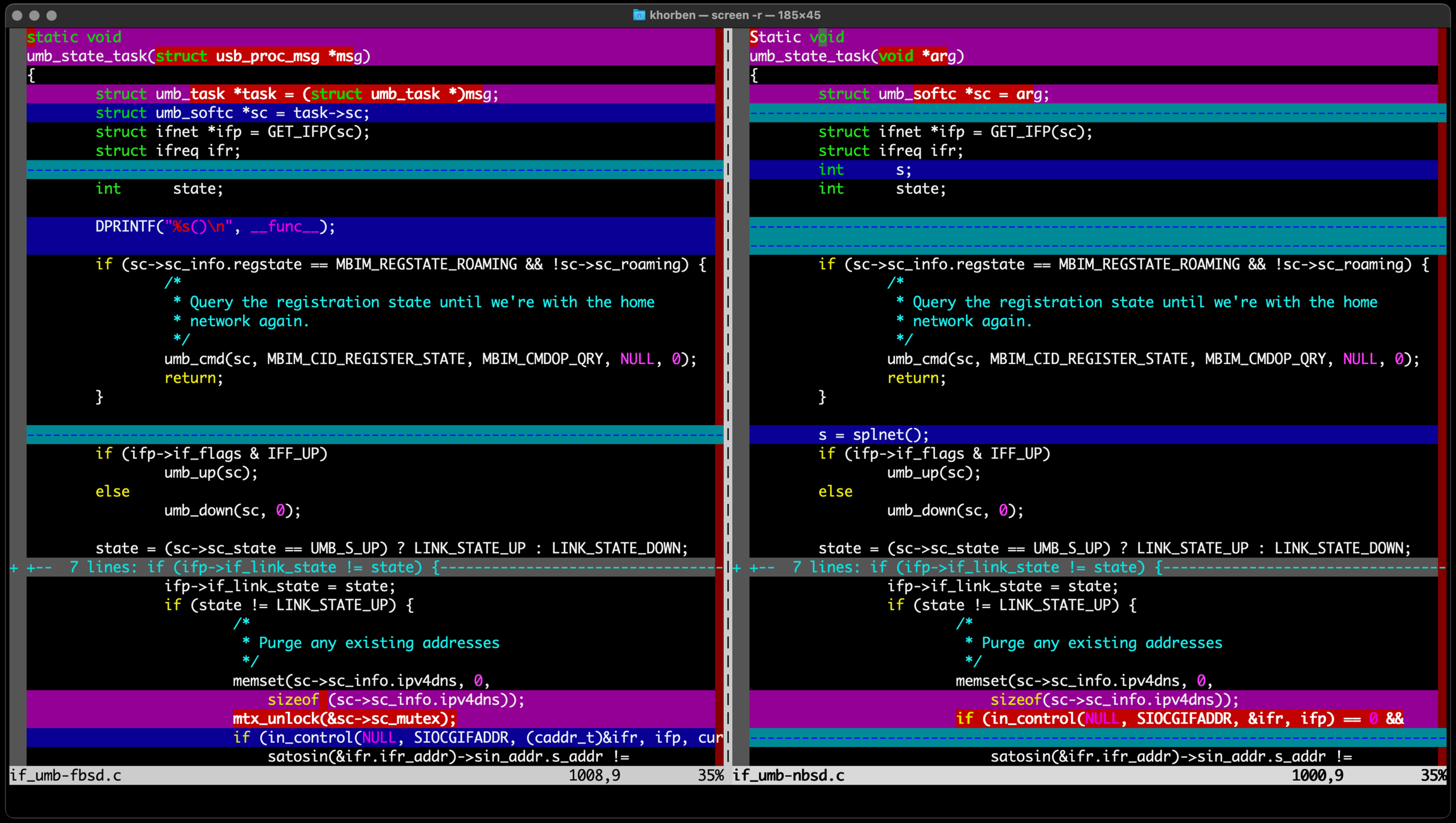

khorben — screen -r — 185×45
sc->sc_if = NULL;
}
return 0;
}
static void
umb_ncm_setup(struct umb_softc *sc, struct usb_config * config)
{
usb_device_request_t req;
struct ncm_ntb_parameters np;
usb_error_t error;
/* Query NTB tranfers sizes */
req.bmRequestType = UT_READ_CLASS_INTERFACE;
req.bRequest = NCM_GET_NTB_PARAMETERS;
USETW(req.wValue, 0);
USETW(req.wIndex, sc->sc_ctrl_ifaceno);
USETW(req.wLength, sizeof (np));
mtx_lock(&sc->sc_mutex);
error = usbd_do_request(sc->sc_udev, &sc->sc_mutex, &req, &np);
mtx_unlock(&sc->sc_mutex);
if_umb-fbsd.c 629,3-17 21%
}
sc->sc_attached = 0;
splx(s);
return 0;
}
Static int
umb_activate(device_t self, enum devact act)
{
struct umb_softc *sc = device_private(self);
switch (act) {
case DVACT_DEACTIVATE:
if_deactivate(GET_IFP(sc));
sc->sc_dying = 1;
return 0;
default:
return EOPNOTSUPP;
}
}
Static void
umb_ncm_setup(struct umb_softc *sc)
{
usb_device_request_t req;
struct ncm_ntb_parameters np;
/* Query NTB tranfers sizes */
req.bmRequestType = UT_READ_CLASS_INTERFACE;
req.bRequest = NCM_GET_NTB_PARAMETERS;
USETW(req.wValue, 0);
USETW(req.wIndex, sc->sc_ctrl_ifaceno);
USETW(req.wLength, sizeof(np));
if (usbd_do_request(sc->sc_udev, &req, &np) == USBD_NORMAL_COMPLETION &&
UGETW(np.wLength) == sizeof(np)) {
sc->sc_rx_bufsz = UGETDW(np.dwNtbInMaxSize);
if_umb-nbsd.c 610,3-10 21%


khorben — screen -r — 185×45
static int
umb_ioctl(struct ifnet *ifp, u_long cmd, caddr_t data)
{
struct umb_softc *sc = ifp->if_softc;
struct in_ifaddr *ia = (struct in_ifaddr *)data;
struct ifreq *ifr = (struct ifreq *)data;
int error = 0;
struct umb_parameter mp;
if (sc->sc_dying)
return EIO;
switch (cmd) {
case SIOCSIFADDR:
switch (ia->ia_ifa.ifa_addr->sa_family) {
case AF_INET:
break;
#ifdef INET6
case AF_INET6:
break;
#endif /* INET6 */
default:
error = EAFNOSUPPORT;
break;
}
break;
case SIOCSIFFLAGS:
mtx_lock(&sc->sc_mutex);
umb_add_task(sc, umb_state_task,
&sc->sc_proc_state_task[0].hdr,
&sc->sc_proc_state_task[1].hdr, 1);
mtx_unlock(&sc->sc_mutex);
break;
if_umb-fbsd.c 743,2-16 24%
Static int
umb_ioctl(struct ifnet *ifp, u_long cmd, void *data)
{
struct umb_softc *sc = ifp->if_softc;
struct ifaddr *ifa = (struct ifaddr *)data;
struct ifreq *ifr = (struct ifreq *)data;
int s, error = 0;
struct umb_parameter mp;
if (sc->sc_dying)
return EIO;
s = splnet();
switch (cmd) {
case SIOCINITIFADDR:
ifp->if_flags |= IFF_UP;
usb_add_task(sc->sc_udev, &sc->sc_umb_task, USB_TASKQ_DRIVER);
switch (ifa->ifa_addr->sa_family) {
#ifdef INET
case AF_INET:
break;
#endif /* INET */
#ifdef INET6
case AF_INET6:
break;
#endif /* INET6 */
default:
error = EAFNOSUPPORT;
break;
}
ifa->ifa_rtrequest = p2p_rtrequest;
break;
case SIOCSIFFLAGS:
error = ifioctl_common(ifp, cmd, data);
if (error)
break;
usb_add_task(sc->sc_udev, &sc->sc_umb_task, USB_TASKQ_DRIVER);
break;
if_umb-nbsd.c 780,2-16 26%


khorben — screen -r — 185×45
mtx_unlock(&sc->sc_mutex);
break;
case SIOCGUMBINFO:
error = copyout(&sc->sc_info, ifr->ifr_ifru.ifru_data,
sizeof (sc->sc_info));
break;
case SIOCSUMBPARAM:
error = priv_check(curthread, PRIV_NET_SETIFPHYS);
if (error)
break;
if ((error = copyin(ifr->ifr_ifru.ifru_data, &mp, sizeof (mp))) !
break;
if ((error = umb_setpin(sc, mp.op, mp.is_puk, mp.pin, mp.pinlen,
mp.newpin, mp.newpinlen)) != 0)
break;
if (mp.apnlen < 0 || mp.apnlen > sizeof (sc->sc_info.apn)) {
error = EINVAL;
break;
}
sc->sc_roaming = mp.roaming ? 1 : 0;
memset(sc->sc_info.apn, 0, sizeof (sc->sc_info.apn));
memcpy(sc->sc_info.apn, mp.apn, mp.apnlen);
sc->sc_info.apnlen = mp.apnlen;
memset(sc->sc_info.username, 0, sizeof (sc->sc_info.username));
memcpy(sc->sc_info.username, mp.username, mp.usernamelen);
sc->sc_info.usernamelen = mp.usernamelen;
memset(sc->sc_info.password, 0, sizeof (sc->sc_info.password));
if_umb-fbsd.c 742,3-17 25%
break;
case SIOCGUMBINFO:
error = kauth_authorize_network(curlwp->l_cred,
KAUTH_NETWORK_INTERFACE,
KAUTH_REQ_NETWORK_INTERFACE_SETPRIV, ifp, KAUTH_ARG(cmd),
NULL);
if (error)
break;
error = copyout(&sc->sc_info, ifr->ifr_data,
sizeof(sc->sc_info));
break;
case SIOCSUMBPARAM:
error = kauth_authorize_network(curlwp->l_cred,
KAUTH_NETWORK_INTERFACE,
KAUTH_REQ_NETWORK_INTERFACE_SETPRIV, ifp, KAUTH_ARG(cmd),
NULL);
if (error)
break;
if ((error = copyin(ifr->ifr_data, &mp, sizeof(mp))) != 0)
break;
if ((error = umb_setpin(sc, mp.op, mp.is_puk, mp.pin, mp.pinlen,
mp.newpin, mp.newpinlen)) != 0)
break;
if (mp.apnlen < 0 || mp.apnlen > sizeof(sc->sc_info.apn)) {
error = EINVAL;
break;
}
sc->sc_roaming = mp.roaming ? 1 : 0;
memset(sc->sc_info.apn, 0, sizeof(sc->sc_info.apn));
memcpy(sc->sc_info.apn, mp.apn, mp.apnlen);
sc->sc_info.apnlen = mp.apnlen;
memset(sc->sc_info.username, 0, sizeof(sc->sc_info.username));
memcpy(sc->sc_info.username, mp.username, mp.usernamelen);
sc->sc_info.usernamelen = mp.usernamelen;
memset(sc->sc_info.password, 0, sizeof(sc->sc_info.password));
if_umb-nbsd.c 780,3-17 27%


khorben — screen -r — 185×45
sc->sc_info.usernamelen = mp.usernamelen;
memset(sc->sc_info.password, 0, sizeof (sc->sc_info.password));
memcpy(sc->sc_info.password, mp.password, mp.passwordlen);
sc->sc_info.passwordlen = mp.passwordlen;
sc->sc_info.preferredclasses = mp.preferredclasses;
umb_setdataclass(sc);
break;
case SIOCGUMBPARAM:
memset(&mp, 0, sizeof (mp));
memcpy(mp.apn, sc->sc_info.apn, sc->sc_info.apnlen);
mp.apnlen = sc->sc_info.apnlen;
mp.roaming = sc->sc_roaming;
mp.preferredclasses = sc->sc_info.preferredclasses;
error = copyout(&mp, ifr->ifr_ifru.ifru_data, sizeof (mp));
break;
case SIOCSIFMTU:
/* Does this include the NCM headers and tail? */
if (ifr->ifr_mtu > ifp->if_mtu) {
error = EINVAL;
break;
}
ifp->if_mtu = ifr->ifr_mtu;
break;
case SIOCAIFADDR:
case SIOCSIFDSTADDR:
case SIOCADDMULTI:
case SIOCDELMULTI:
break;
case SIOCGIFMEDIA:
error = ifmedia_ioctl(ifp, ifr, &sc->sc_im, cmd);
break;
default:
error = EINVAL;
break;
}
return (error);
}
if_umb-fbsd.c 770,3-17 26%
sc->sc_info.usernamelen = mp.usernamelen;
memset(sc->sc_info.password, 0, sizeof(sc->sc_info.password));
memcpy(sc->sc_info.password, mp.password, mp.passwordlen);
sc->sc_info.passwordlen = mp.passwordlen;
sc->sc_info.preferredclasses = mp.preferredclasses;
umb_setdataclass(sc);
break;
case SIOCGUMBPARAM:
memset(&mp, 0, sizeof(mp));
memcpy(mp.apn, sc->sc_info.apn, sc->sc_info.apnlen);
mp.apnlen = sc->sc_info.apnlen;
mp.roaming = sc->sc_roaming;
mp.preferredclasses = sc->sc_info.preferredclasses;
error = copyout(&mp, ifr->ifr_data, sizeof(mp));
break;
case SIOCSIFMTU:
/* Does this include the NCM headers and tail? */
if (ifr->ifr_mtu > ifp->if_mtu) {
error = EINVAL;
break;
}
ifp->if_mtu = ifr->ifr_mtu;
break;
case SIOCSIFADDR:
case SIOCAIFADDR:
case SIOCSIFDSTADDR:
case SIOCADDMULTI:
case SIOCDELMULTI:
break;
case SIOCGIFMEDIA:
error = ifmedia_ioctl(ifp, ifr, &sc->sc_im, cmd);
break;
default:
error = ifioctl_common(ifp, cmd, data);
break;
}
splx(s);
return error;
if_umb-nbsd.c 816,3-17 28%


khorben — screen -r — 185×45
mtx_assert(&sc->sc_mutex, MA_OWNED);
if (usb_proc_is_gone(&sc->sc_taskqueue)) {
return;
}
task = usb_proc_msignal(&sc->sc_taskqueue, t0, t1);
task->hdr.pm_callback = callback;
task->sc = sc;
if (sync) {
usb_proc_mwait(&sc->sc_taskqueue, t0, t1);
}
}
static void
umb_newstate(struct umb_softc *sc, enum umb_state newstate, int flags)
{
struct ifnet *ifp = GET_IFP(sc);
if (newstate == sc->sc_state)
return;
+-- 2 lines: if (((flags & UMB_NS_DONT_DROP) && newstate < sc->sc_state) ||-----
return;
if (ifp->if_flags & IFF_DEBUG)
log(LOG_DEBUG, "%s: state going %s from '%s' to '%s'\n",
DEVNAM(sc), newstate > sc->sc_state ? "up" : "down",
umb_istate(sc->sc_state), umb_istate(newstate));
sc->sc_state = newstate;
mtx_lock(&sc->sc_mutex);
umb_add_task(sc, umb_state_task,
&sc->sc_proc_state_task[0].hdr,
&sc->sc_proc_state_task[1].hdr, 0);
mtx_unlock(&sc->sc_mutex);
}
static void
if_umb-fbsd.c 1008,1 33%
umb_newstate(struct umb_softc *sc, enum umb_state newstate, int flags)
{
struct ifnet *ifp = GET_IFP(sc);
if (newstate == sc->sc_state)
return;
+-- 2 lines: if (((flags & UMB_NS_DONT_DROP) && newstate < sc->sc_state) ||-----
return;
if (ifp->if_flags & IFF_DEBUG)
log(LOG_DEBUG, "%s: state going %s from '%s' to '%s'\n",
DEVNAM(sc), newstate > sc->sc_state ? "up" : "down",
umb_istate(sc->sc_state), umb_istate(newstate));
sc->sc_state = newstate;
usb_add_task(sc->sc_udev, &sc->sc_umb_task, USB_TASKQ_DRIVER);
}
Static void
if_umb-nbsd.c 1000,1 34%


khorben — screen -r — 185×45
static void
umb_state_task(struct usb_proc_msg *msg)
{
struct umb_task *task = (struct umb_task *)msg;
struct umb_softc *sc = task->sc;
struct ifnet *ifp = GET_IFP(sc);
struct ifreq ifr;
int state;
DPRINTF("%s()\n", __func__);
if (sc->sc_info.regstate == MBIM_REGSTATE_ROAMING && !sc->sc_roaming) {
/*
* Query the registration state until we're with the home
* network again.
*/
umb_cmd(sc, MBIM_CID_REGISTER_STATE, MBIM_CMDOP_QRY, NULL, 0);
return;
}
if (ifp->if_flags & IFF_UP)
umb_up(sc);
else
umb_down(sc, 0);
state = (sc->sc_state == UMB_S_UP) ? LINK_STATE_UP : LINK_STATE_DOWN;
+-- 7 lines: if (ifp->if_link_state != state) {
ifp->if_link_state = state;
if (state != LINK_STATE_UP) {
/*
* Purge any existing addresses
*/
memset(sc->sc_info.ipv4dns, 0,
sizeof (sc->sc_info.ipv4dns));
mtx_unlock(&sc->sc_mutex);
if (in_control(NULL, SIOCGIFADDR, (caddr_t)&ifr, ifp, cur
satosin(&ifr.ifr_addr)->sin_addr.s_addr !=
if_umb-fbsd.c 1008,9 35%
Static void
umb_state_task(void *arg)
{
struct umb_softc *sc = arg;
struct ifnet *ifp = GET_IFP(sc);
struct ifreq ifr;
int s;
int state;
if (sc->sc_info.regstate == MBIM_REGSTATE_ROAMING && !sc->sc_roaming) {
/*
* Query the registration state until we're with the home
* network again.
*/
umb_cmd(sc, MBIM_CID_REGISTER_STATE, MBIM_CMDOP_QRY, NULL, 0);
return;
}
s = splnet();
if (ifp->if_flags & IFF_UP)
umb_up(sc);
else
umb_down(sc, 0);
state = (sc->sc_state == UMB_S_UP) ? LINK_STATE_UP : LINK_STATE_DOWN;
+-- 7 lines: if (ifp->if_link_state != state) {
ifp->if_link_state = state;
if (state != LINK_STATE_UP) {
/*
* Purge any existing addresses
*/
memset(sc->sc_info.ipv4dns, 0,
sizeof(sc->sc_info.ipv4dns));
if (in_control(NULL, SIOCGIFADDR, &ifr, ifp) == 0 &&
satosin(&ifr.ifr_addr)->sin_addr.s_addr !=
if_umb-nbsd.c 1000,9 35%

You Get The Idea

Dream vs Reality

The dream: one driver API for BSD!

The reality: no chance?

Would we not all benefit?

Alternate strategies:

- Coordinate
- Take steps to get closer?
  - Communication
  - Driver code

You Get The Idea

Some Possibilities: Driver Code

Different things can be done:

- Separate files:
  - Constant variables
  - BSD-specific vs driver-specific code
- Abstraction layer:
  - Function prototypes
  - Variable types
  - Subsystem access…

You Get The Idea

Some Possibilities: Beyond Driver Code

System databases:

- PCI & USB IDs
- Register names & values
- Driver names

Methodology:

- Sharing Git (or Got) commits

You Get The Idea

Some Possibilities: Communication

Cross-BSD committee:

- Mailing-list: bsd-drivers@lists.EdgeBSD.org
  - Instructions: https://lists.EdgeBSD.org/#bsd-drivers
  - Archives: https://lists.EdgeBSD.org/bsd-drivers/
- IRC channel, regular meetings?
- Documentation? (Driver database, wiki…)
- Funding?

Conclusion
BSD Driver Harmony

- Each BSD is its own system & community
  - Userland code is very similar (POSIX!)
  - Kernel drivers are more challenging but remain close
- Call for Action!
- Worth the effort?
- Like, comment, subscribe:
  Pierre Pronchery <khorben@NetBSD.org>
  On the NetBSD Foundation's Board <board@NetBSD.org>