|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Cruz Carlon Juan Alfredo M.C. |
| *Asignatura:* | Fundamentos de programación |
| *Grupo:* | 1107 |
| *N° de Práctica(s):* | Práctica 12 |
| *Integrante(s):* | Brenda Juárez, Eduardo Norman, Fátima, e Irving Silverio |
| *Semestre:* | 2018-I |
| *Fecha de entrega:* | 28-Noviembre-2017 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

Funciones: “todo programa es un función pero no toda función es programa”

¿por qué no toda función es un programa?

Porque un programa es la resolución de un problema complejo y a veces largo el cual su resolución se divide en pequeños bloques o funciones correlacionadas que calculan una parte del resultado final en el cual se ven involucradas las variables comunes, mientras que un función es simplemente un bloque compuesto con cabecera y variables parciales que nos proporcionan solo una parte del resultado final deseado.

#include <cache.h>

#undef DEBUG\_85

#ifdef DEBUG\_85

#define say(a) fprintf(stderr, a)

#define say1(a,b) fprintf(stderr, a, b)

#define say2(a,b,c) fprintf(stderr, a, b, c)

#else

#define say(a) do { /\* nothing \*/ } while (0)

#define say1(a,b) do { /\* nothing \*/ } while (0)

#define say2(a,b,c) do { /\* nothing \*/ } while (0)

#endif

static const char en85[] = {

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',

'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',

'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',

'U', 'V', 'W', 'X', 'Y', 'Z',

'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',

'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',

'u', 'v', 'w', 'x', 'y', 'z',

'!', '#', '$', '%', '&', '(', ')', '\*', '+', '-',

';', '<', '=', '>', '?', '@', '^', '\_', '`', '{',

'|', '}', '~'

};

static char de85[256];

static void prep\_base85(void)

{

int i;

if (de85['Z'])

return;

for (i = 0; i < ARRAY\_SIZE(en85); i++) {

int ch = en85[i];

de85[ch] = i + 1;

}

}

int decode\_85(char \*dst, const char \*buffer, int len)

{

prep\_base85();

say2("decode 85 <%.\*s>", len / 4 \* 5, buffer);

while (len) {

unsigned acc = 0;

int de, cnt = 4;

unsigned char ch;

do {

ch = \*buffer++;

de = de85[ch];

if (--de < 0)

return error("invalid base85 alphabet %c", ch);

acc = acc \* 85 + de;

} while (--cnt);

ch = \*buffer++;

de = de85[ch];

if (--de < 0)

return error("invalid base85 alphabet %c", ch);

/\* Detect overflow. \*/

if (0xffffffff / 85 < acc ||

0xffffffff - de < (acc \*= 85))

return error("invalid base85 sequence %.5s", buffer-5);

acc += de;

say1(" %08x", acc);

cnt = (len < 4) ? len : 4;

len -= cnt;

do {

acc = (acc << 8) | (acc >> 24);

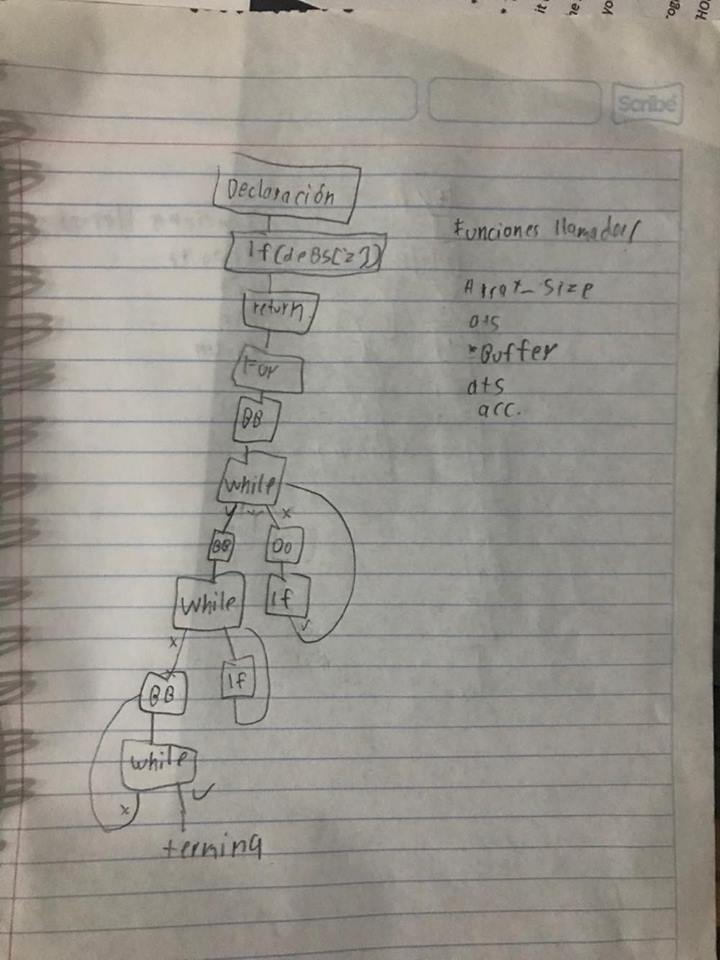
\*dst++ = acc;

} while (--cnt);

}

say("\n");

return 0;



}

void encode\_85(char \*buf, const unsigned char \*data, int bytes)

{

say("encode 85");

while (bytes) {

unsigned acc = 0;

int cnt;

for (cnt = 24; cnt >= 0; cnt -= 8) {

unsigned ch = \*data++;

acc |= ch << cnt;

if (--bytes == 0)

break;

}

say1(" %08x", acc);

for (cnt = 4; cnt >= 0; cnt--) {

int val = acc % 85;

acc /= 85;

buf[cnt] = en85[val];

}

buf += 5;

}

say("\n");

\*buf = 0;

}

#ifdef DEBUG\_85

int main(int ac, char \*\*av)

{

char buf[1024];

if (!strcmp(av[1], "-e")) {

int len = strlen(av[2]);

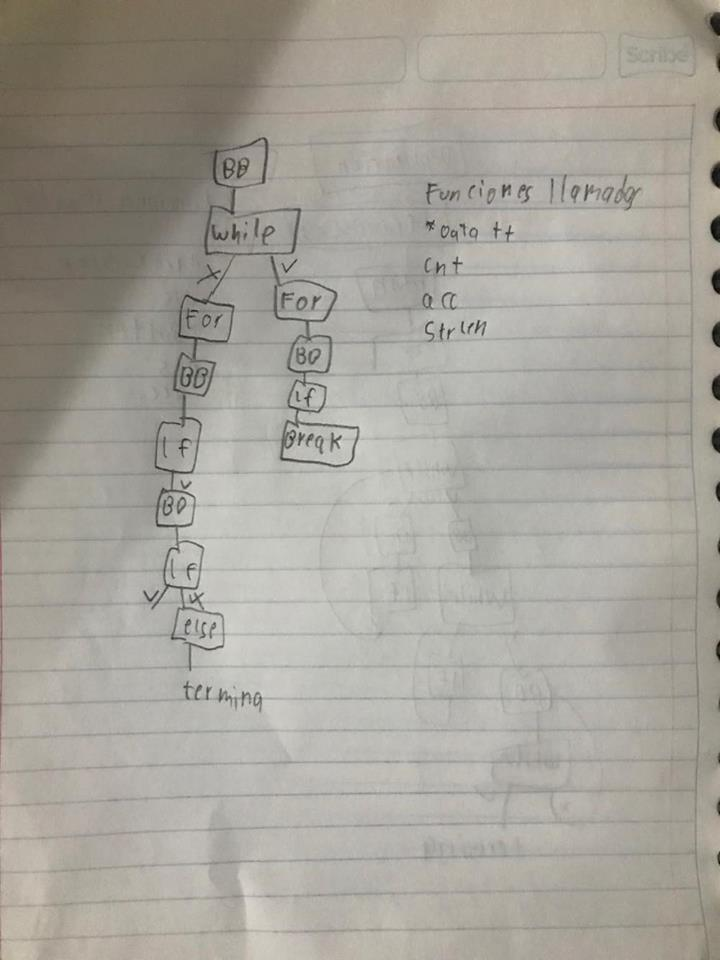
encode\_85(buf, av[2], len);

if (len <= 26) len = len + 'A' - 1;

else len = len + 'a' - 26 - 1;

printf("encoded: %c%s\n", len, buf);

return 0;



}

if (!strcmp(av[1], "-d")) {

int len = \*av[2];

if ('A' <= len && len <= 'Z') len = len - 'A' + 1;

else len = len - 'a' + 26 + 1;

decode\_85(buf, av[2]+1, len);

printf("decoded: %.\*s\n", len, buf);

return 0;

}

if (!strcmp(av[1], "-t")) {

char t[4] = { -1,-1,-1,-1 };

encode\_85(buf, t, 4);

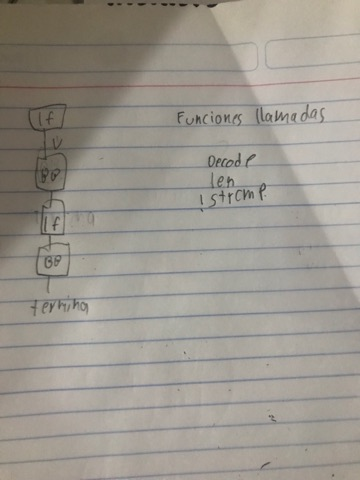
printf("encoded: D%s\n", buf);

return 0;

}

}

#endif



/\*

\*

\* Bluetooth support for Broadcom devices

\*

\* Copyright (C) 2015 Intel Corporation

\*

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation; either version 2 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program; if not, write to the Free Software

\* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*

\*/

#include <linux/module.h>

#include <linux/firmware.h>

#include <asm/unaligned.h>

#include <net/bluetooth/bluetooth.h>

#include <net/bluetooth/hci\_core.h>

#include "btbcm.h"

#define VERSION "0.1"

#define BDADDR\_BCM20702A0 (&(bdaddr\_t) {{0x00, 0xa0, 0x02, 0x70, 0x20, 0x00}})

#define BDADDR\_BCM4324B3 (&(bdaddr\_t) {{0x00, 0x00, 0x00, 0xb3, 0x24, 0x43}})

#define BDADDR\_BCM4330B1 (&(bdaddr\_t) {{0x00, 0x00, 0x00, 0xb1, 0x30, 0x43}})

int btbcm\_check\_bdaddr(struct hci\_dev \*hdev)

{

struct hci\_rp\_read\_bd\_addr \*bda;

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, HCI\_OP\_READ\_BD\_ADDR, 0, NULL,

HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

int err = PTR\_ERR(skb);

bt\_dev\_err(hdev, "BCM: Reading device address failed (%d)", err);

return err;

}

if (skb->len != sizeof(\*bda)) {

bt\_dev\_err(hdev, "BCM: Device address length mismatch");

kfree\_skb(skb);

return -EIO;

}

bda = (struct hci\_rp\_read\_bd\_addr \*)skb->data;

/\* Check if the address indicates a controller with either an

\* invalid or default address. In both cases the device needs

\* to be marked as not having a valid address.

\*

\* The address 00:20:70:02:A0:00 indicates a BCM20702A0 controller

\* with no configured address.

\*

\* The address 43:24:B3:00:00:00 indicates a BCM4324B3 controller

\* with waiting for configuration state.

\*

\* The address 43:30:B1:00:00:00 indicates a BCM4330B1 controller

\* with waiting for configuration state.

\*/

if (!bacmp(&bda->bdaddr, BDADDR\_BCM20702A0) ||

!bacmp(&bda->bdaddr, BDADDR\_BCM4324B3) ||

!bacmp(&bda->bdaddr, BDADDR\_BCM4330B1)) {

bt\_dev\_info(hdev, "BCM: Using default device address (%pMR)",

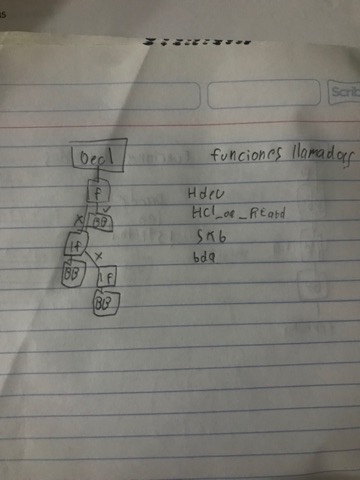
&bda->bdaddr);

set\_bit(HCI\_QUIRK\_INVALID\_BDADDR, &hdev->quirks);

}

kfree\_skb(skb);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_check\_bdaddr);

int btbcm\_set\_bdaddr(struct hci\_dev \*hdev, const bdaddr\_t \*bdaddr)

{

struct sk\_buff \*skb;

int err;

skb = \_\_hci\_cmd\_sync(hdev, 0xfc01, 6, bdaddr, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

err = PTR\_ERR(skb);

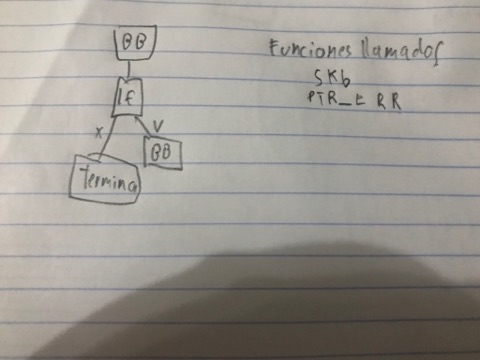
bt\_dev\_err(hdev, "BCM: Change address command failed (%d)", err);

return err;

}

kfree\_skb(skb);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_set\_bdaddr);

int btbcm\_patchram(struct hci\_dev \*hdev, const struct firmware \*fw)

{

const struct hci\_command\_hdr \*cmd;

const u8 \*fw\_ptr;

size\_t fw\_size;

struct sk\_buff \*skb;

u16 opcode;

int err = 0;

/\* Start Download \*/

skb = \_\_hci\_cmd\_sync(hdev, 0xfc2e, 0, NULL, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

err = PTR\_ERR(skb);

bt\_dev\_err(hdev, "BCM: Download Minidrv command failed (%d)",

err);

goto done;

}

kfree\_skb(skb);

/\* 50 msec delay after Download Minidrv completes \*/

msleep(50);

fw\_ptr = fw->data;

fw\_size = fw->size;

while (fw\_size >= sizeof(\*cmd)) {

const u8 \*cmd\_param;

cmd = (struct hci\_command\_hdr \*)fw\_ptr;

fw\_ptr += sizeof(\*cmd);

fw\_size -= sizeof(\*cmd);

if (fw\_size < cmd->plen) {

bt\_dev\_err(hdev, "BCM: Patch is corrupted");

err = -EINVAL;

goto done;

}

cmd\_param = fw\_ptr;

fw\_ptr += cmd->plen;

fw\_size -= cmd->plen;

opcode = le16\_to\_cpu(cmd->opcode);

skb = \_\_hci\_cmd\_sync(hdev, opcode, cmd->plen, cmd\_param,

HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

err = PTR\_ERR(skb);

bt\_dev\_err(hdev, "BCM: Patch command %04x failed (%d)",

opcode, err);

goto done;

}

kfree\_skb(skb);

}

/\* 250 msec delay after Launch Ram completes \*/

msleep(250);

done:

return err;

}

EXPORT\_SYMBOL(btbcm\_patchram);

static int btbcm\_reset(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, HCI\_OP\_RESET, 0, NULL, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

int err = PTR\_ERR(skb);

bt\_dev\_err(hdev, "BCM: Reset failed (%d)", err);

return err;

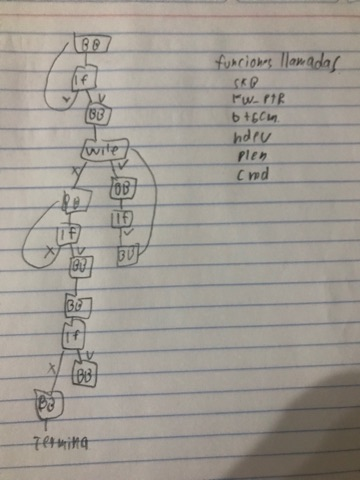
}

kfree\_skb(skb);

/\* 100 msec delay for module to complete reset process \*/

msleep(100);

return 0;



}

static struct sk\_buff \*btbcm\_read\_local\_name(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, HCI\_OP\_READ\_LOCAL\_NAME, 0, NULL,

HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

bt\_dev\_err(hdev, "BCM: Reading local name failed (%ld)",

PTR\_ERR(skb));

return skb;

}

if (skb->len != sizeof(struct hci\_rp\_read\_local\_name)) {

bt\_dev\_err(hdev, "BCM: Local name length mismatch");

kfree\_skb(skb);

return ERR\_PTR(-EIO);

}

return skb;

}

static struct sk\_buff \*btbcm\_read\_local\_version(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, HCI\_OP\_READ\_LOCAL\_VERSION, 0, NULL,

HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

bt\_dev\_err(hdev, "BCM: Reading local version info failed (%ld)",

PTR\_ERR(skb));

return skb;

}

if (skb->len != sizeof(struct hci\_rp\_read\_local\_version)) {

bt\_dev\_err(hdev, "BCM: Local version length mismatch");

kfree\_skb(skb);

return ERR\_PTR(-EIO);

}

return skb;

}

static struct sk\_buff \*btbcm\_read\_verbose\_config(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, 0xfc79, 0, NULL, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

bt\_dev\_err(hdev, "BCM: Read verbose config info failed (%ld)",

PTR\_ERR(skb));

return skb;

}

if (skb->len != 7) {

bt\_dev\_err(hdev, "BCM: Verbose config length mismatch");

kfree\_skb(skb);

return ERR\_PTR(-EIO);

}

return skb;

}

static struct sk\_buff \*btbcm\_read\_controller\_features(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, 0xfc6e, 0, NULL, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

bt\_dev\_err(hdev, "BCM: Read controller features failed (%ld)",

PTR\_ERR(skb));

return skb;

}

if (skb->len != 9) {

bt\_dev\_err(hdev, "BCM: Controller features length mismatch");

kfree\_skb(skb);

return ERR\_PTR(-EIO);

}

return skb;

}

static struct sk\_buff \*btbcm\_read\_usb\_product(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

skb = \_\_hci\_cmd\_sync(hdev, 0xfc5a, 0, NULL, HCI\_INIT\_TIMEOUT);

if (IS\_ERR(skb)) {

bt\_dev\_err(hdev, "BCM: Read USB product info failed (%ld)",

PTR\_ERR(skb));

return skb;

}

if (skb->len != 5) {

bt\_dev\_err(hdev, "BCM: USB product length mismatch");

kfree\_skb(skb);

return ERR\_PTR(-EIO);

}

return skb;

}

static int btbcm\_read\_info(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

/\* Read Verbose Config Version Info \*/

skb = btbcm\_read\_verbose\_config(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

bt\_dev\_info(hdev, "BCM: chip id %u", skb->data[1]);

kfree\_skb(skb);

/\* Read Controller Features \*/

skb = btbcm\_read\_controller\_features(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

bt\_dev\_info(hdev, "BCM: features 0x%2.2x", skb->data[1]);

kfree\_skb(skb);

/\* Read Local Name \*/

skb = btbcm\_read\_local\_name(hdev);

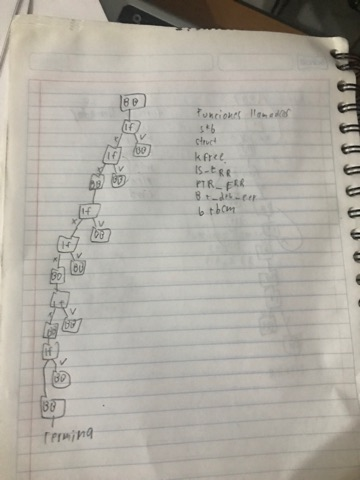
if (IS\_ERR(skb))

return PTR\_ERR(skb);

bt\_dev\_info(hdev, "%s", (char \*)(skb->data + 1));

kfree\_skb(skb);

return 0;



}

static const struct {

u16 subver;

const char \*name;

} bcm\_uart\_subver\_table[] = {

{ 0x4103, "BCM4330B1" }, /\* 002.001.003 \*/

{ 0x410e, "BCM43341B0" }, /\* 002.001.014 \*/

{ 0x4406, "BCM4324B3" }, /\* 002.004.006 \*/

{ 0x610c, "BCM4354" }, /\* 003.001.012 \*/

{ 0x2209, "BCM43430A1" }, /\* 001.002.009 \*/

{ 0x6119, "BCM4345C0" }, /\* 003.001.025 \*/

{ 0x230f, "BCM4356A2" }, /\* 001.003.015 \*/

{ }

};

int btbcm\_initialize(struct hci\_dev \*hdev, char \*fw\_name, size\_t len)

{

u16 subver, rev;

const char \*hw\_name = NULL;

struct sk\_buff \*skb;

struct hci\_rp\_read\_local\_version \*ver;

int i, err;

/\* Reset \*/

err = btbcm\_reset(hdev);

if (err)

return err;

/\* Read Local Version Info \*/

skb = btbcm\_read\_local\_version(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

ver = (struct hci\_rp\_read\_local\_version \*)skb->data;

rev = le16\_to\_cpu(ver->hci\_rev);

subver = le16\_to\_cpu(ver->lmp\_subver);

kfree\_skb(skb);

/\* Read controller information \*/

err = btbcm\_read\_info(hdev);

if (err)

return err;

switch ((rev & 0xf000) >> 12) {

case 0:

case 1:

case 2:

case 3:

for (i = 0; bcm\_uart\_subver\_table[i].name; i++) {

if (subver == bcm\_uart\_subver\_table[i].subver) {

hw\_name = bcm\_uart\_subver\_table[i].name;

break;

}

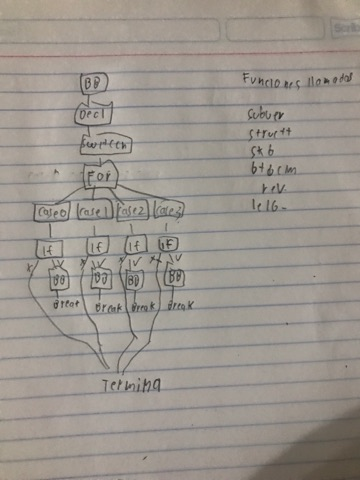
}

snprintf(fw\_name, len, "brcm/%s.hcd", hw\_name ? : "BCM");

break;

default:

return 0;



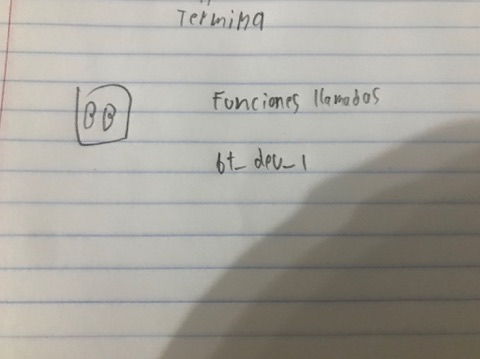
}

bt\_dev\_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",

hw\_name ? : "BCM", (subver & 0xe000) >> 13,

(subver & 0x1f00) >> 8, (subver & 0x00ff), rev & 0x0fff);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_initialize);

int btbcm\_finalize(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

struct hci\_rp\_read\_local\_version \*ver;

u16 subver, rev;

int err;

/\* Reset \*/

err = btbcm\_reset(hdev);

if (err)

return err;

/\* Read Local Version Info \*/

skb = btbcm\_read\_local\_version(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

ver = (struct hci\_rp\_read\_local\_version \*)skb->data;

rev = le16\_to\_cpu(ver->hci\_rev);

subver = le16\_to\_cpu(ver->lmp\_subver);

kfree\_skb(skb);

bt\_dev\_info(hdev, "BCM (%3.3u.%3.3u.%3.3u) build %4.4u",

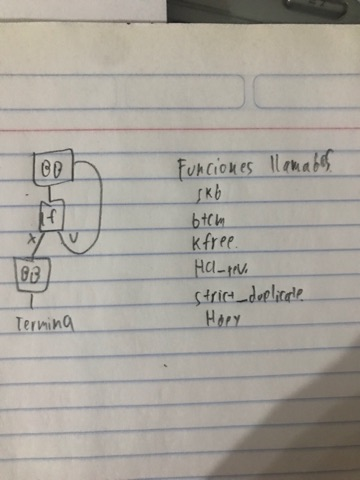
(subver & 0xe000) >> 13, (subver & 0x1f00) >> 8,

(subver & 0x00ff), rev & 0x0fff);

btbcm\_check\_bdaddr(hdev);

set\_bit(HCI\_QUIRK\_STRICT\_DUPLICATE\_FILTER, &hdev->quirks);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_finalize);

static const struct {

u16 subver;

const char \*name;

} bcm\_usb\_subver\_table[] = {

{ 0x210b, "BCM43142A0" }, /\* 001.001.011 \*/

{ 0x2112, "BCM4314A0" }, /\* 001.001.018 \*/

{ 0x2118, "BCM20702A0" }, /\* 001.001.024 \*/

{ 0x2126, "BCM4335A0" }, /\* 001.001.038 \*/

{ 0x220e, "BCM20702A1" }, /\* 001.002.014 \*/

{ 0x230f, "BCM4354A2" }, /\* 001.003.015 \*/

{ 0x4106, "BCM4335B0" }, /\* 002.001.006 \*/

{ 0x410e, "BCM20702B0" }, /\* 002.001.014 \*/

{ 0x6109, "BCM4335C0" }, /\* 003.001.009 \*/

{ 0x610c, "BCM4354" }, /\* 003.001.012 \*/

{ }

};

int btbcm\_setup\_patchram(struct hci\_dev \*hdev)

{

char fw\_name[64];

const struct firmware \*fw;

u16 subver, rev, pid, vid;

const char \*hw\_name = NULL;

struct sk\_buff \*skb;

struct hci\_rp\_read\_local\_version \*ver;

int i, err;

/\* Reset \*/

err = btbcm\_reset(hdev);

if (err)

return err;

/\* Read Local Version Info \*/

skb = btbcm\_read\_local\_version(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

ver = (struct hci\_rp\_read\_local\_version \*)skb->data;

rev = le16\_to\_cpu(ver->hci\_rev);

subver = le16\_to\_cpu(ver->lmp\_subver);

kfree\_skb(skb);

/\* Read controller information \*/

err = btbcm\_read\_info(hdev);

if (err)

return err;

switch ((rev & 0xf000) >> 12) {

case 0:

case 3:

for (i = 0; bcm\_uart\_subver\_table[i].name; i++) {

if (subver == bcm\_uart\_subver\_table[i].subver) {

hw\_name = bcm\_uart\_subver\_table[i].name;

break;

}

}

snprintf(fw\_name, sizeof(fw\_name), "brcm/%s.hcd",

hw\_name ? : "BCM");

break;

case 1:

case 2:

/\* Read USB Product Info \*/

skb = btbcm\_read\_usb\_product(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

vid = get\_unaligned\_le16(skb->data + 1);

pid = get\_unaligned\_le16(skb->data + 3);

kfree\_skb(skb);

for (i = 0; bcm\_usb\_subver\_table[i].name; i++) {

if (subver == bcm\_usb\_subver\_table[i].subver) {

hw\_name = bcm\_usb\_subver\_table[i].name;

break;

}

}

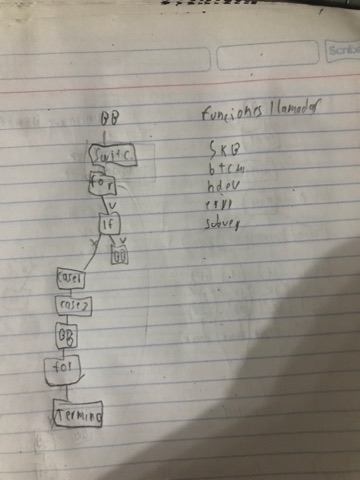
snprintf(fw\_name, sizeof(fw\_name), "brcm/%s-%4.4x-%4.4x.hcd",

hw\_name ? : "BCM", vid, pid);

break;

default:

return 0;



}

bt\_dev\_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",

hw\_name ? : "BCM", (subver & 0xe000) >> 13,

(subver & 0x1f00) >> 8, (subver & 0x00ff), rev & 0x0fff);

err = request\_firmware(&fw, fw\_name, &hdev->dev);

if (err < 0) {

bt\_dev\_info(hdev, "BCM: Patch %s not found", fw\_name);

goto done;

}

btbcm\_patchram(hdev, fw);

release\_firmware(fw);

/\* Reset \*/

err = btbcm\_reset(hdev);

if (err)

return err;

/\* Read Local Version Info \*/

skb = btbcm\_read\_local\_version(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

ver = (struct hci\_rp\_read\_local\_version \*)skb->data;

rev = le16\_to\_cpu(ver->hci\_rev);

subver = le16\_to\_cpu(ver->lmp\_subver);

kfree\_skb(skb);

bt\_dev\_info(hdev, "%s (%3.3u.%3.3u.%3.3u) build %4.4u",

hw\_name ? : "BCM", (subver & 0xe000) >> 13,

(subver & 0x1f00) >> 8, (subver & 0x00ff), rev & 0x0fff);

/\* Read Local Name \*/

skb = btbcm\_read\_local\_name(hdev);

if (IS\_ERR(skb))

return PTR\_ERR(skb);

bt\_dev\_info(hdev, "%s", (char \*)(skb->data + 1));

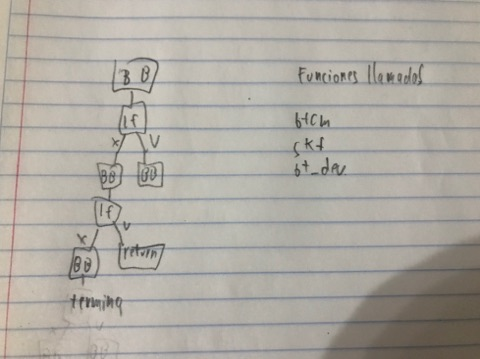
kfree\_skb(skb);

done:

btbcm\_check\_bdaddr(hdev);

set\_bit(HCI\_QUIRK\_STRICT\_DUPLICATE\_FILTER, &hdev->quirks);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_setup\_patchram);

int btbcm\_setup\_apple(struct hci\_dev \*hdev)

{

struct sk\_buff \*skb;

int err;

/\* Reset \*/

err = btbcm\_reset(hdev);

if (err)

return err;

/\* Read Verbose Config Version Info \*/

skb = btbcm\_read\_verbose\_config(hdev);

if (!IS\_ERR(skb)) {

bt\_dev\_info(hdev, "BCM: chip id %u build %4.4u",

skb->data[1], get\_unaligned\_le16(skb->data + 5));

kfree\_skb(skb);

}

/\* Read USB Product Info \*/

skb = btbcm\_read\_usb\_product(hdev);

if (!IS\_ERR(skb)) {

bt\_dev\_info(hdev, "BCM: product %4.4x:%4.4x",

get\_unaligned\_le16(skb->data + 1),

get\_unaligned\_le16(skb->data + 3));

kfree\_skb(skb);

}

/\* Read Controller Features \*/

skb = btbcm\_read\_controller\_features(hdev);

if (!IS\_ERR(skb)) {

bt\_dev\_info(hdev, "BCM: features 0x%2.2x", skb->data[1]);

kfree\_skb(skb);

}

/\* Read Local Name \*/

skb = btbcm\_read\_local\_name(hdev);

if (!IS\_ERR(skb)) {

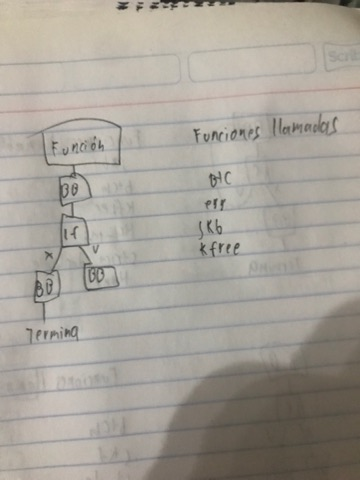
bt\_dev\_info(hdev, "%s", (char \*)(skb->data + 1));

kfree\_skb(skb);

}

set\_bit(HCI\_QUIRK\_STRICT\_DUPLICATE\_FILTER, &hdev->quirks);

return 0;



}

EXPORT\_SYMBOL\_GPL(btbcm\_setup\_apple);

MODULE\_AUTHOR("Marcel Holtmann <marcel@holtmann.org>");

MODULE\_DESCRIPTION("Bluetooth support for Broadcom devices ver " VERSION);

MODULE\_VERSION(VERSION);

MODULE\_LICENSE("GPL");