#include <stdio.h>

#include <unistd.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <mmi\_window.h>

#include <mmi\_module\_manage.h>

#include "mmi\_utils.h"

#include "mmi\_config.h"

#include <hash\_map>

#include <pthread.h>

#include <errno.h>

#include <sys/wait.h>

#include <hardware/hardware.h>

#include <hardware/bluetooth.h>

#include <private/android\_filesystem\_config.h>

#include <sys/prctl.h>

#include <sys/capability.h>

static hash\_map < string, string > paras;

static mmi\_module \*g\_module;

static int g\_retval;

sem\_t g\_sem;

#define TINNO\_MODE

static struct input\_params input;

static pthread\_t processThreadPid;

static class mmi\_text \*text\_waiting;

int module\_main(mmi\_module \* mod);

void pass(void \*);

void fail(void \*);

extern "C" void \_\_attribute\_\_ ((constructor)) register\_module(void);

void register\_module(void) {

g\_module = mmi\_module::register\_module(LOG\_TAG, module\_main);

}

void signalHandler(int signal) {

pthread\_exit(NULL);

}

#define BT\_CONFIG\_OLD "/data/misc/bluedroid/bt\_config.old"

#define BT\_CONFIG\_NEW "/data/misc/bluedroid/bt\_config.xml"

mmi\_text \*text[11];

static bool exit\_thd = false;

static bool bt\_init\_over = false;

int bt\_init\_complete = -1;

static int bt\_enabled=0;

static pthread\_t btThread;

#define CASE\_RETURN\_STR(const) case const: return #const;

static bt\_status\_t status;

static gid\_t groups[] = { AID\_NET\_BT, AID\_INET, AID\_NET\_BT\_ADMIN,

AID\_SYSTEM, AID\_MISC, AID\_SDCARD\_RW,

AID\_NET\_ADMIN, AID\_VPN};

/\* Main API \*/

static bluetooth\_device\_t\* bt\_device;

static const bt\_interface\_t\* sBtInterface = NULL;

bool discovery\_state=false;

bool Scanning\_state=false;

typedef struct \_BT\_Info{

struct \_BT\_Info \*pNext;

uint8\_t bd\_addr[6];

unsigned char psr;

unsigned char cod[3];

unsigned char clkoffset[2];

int32\_t rssi;

char bd\_name[248];

} BT\_Info;

BT\_Info \*g\_pBtListHear = NULL;

BT\_Info \*pTmpBtInfo = NULL,\*mTmpBtInfoForList = NULL;

static const char\* dump\_bt\_status(bt\_status\_t status)

{

switch(status)

{

CASE\_RETURN\_STR(BT\_STATUS\_SUCCESS)

CASE\_RETURN\_STR(BT\_STATUS\_FAIL)

CASE\_RETURN\_STR(BT\_STATUS\_NOT\_READY)

CASE\_RETURN\_STR(BT\_STATUS\_NOMEM)

CASE\_RETURN\_STR(BT\_STATUS\_BUSY)

CASE\_RETURN\_STR(BT\_STATUS\_UNSUPPORTED)

default:

return "unknown status code";

}

}

void bt\_cleanup(void)

{

sBtInterface->cleanup();

}

int HAL\_load(void)

{

int err = 0;

hw\_module\_t\* module;

hw\_device\_t\* device;

err = hw\_get\_module(BT\_HARDWARE\_MODULE\_ID, (hw\_module\_t const\*\*)&module);

if (err == 0)

{

err = module->methods->open(module, BT\_HARDWARE\_MODULE\_ID, &device);

if (err == 0) {

bt\_device = (bluetooth\_device\_t \*)device;

sBtInterface = bt\_device->get\_bluetooth\_interface();

}

}

return err;

}

int HAL\_unload(void)

{

int err = 0;

ALOGE("Unloading HAL lib");

sBtInterface = NULL;

ALOGE("HAL library unloaded (%s)", strerror(err));

return err;

}

void check\_return\_status(bt\_status\_t status)

{

if (status != BT\_STATUS\_SUCCESS)

{

ALOGE("HAL REQUEST FAILED status : %d (%s)", status, dump\_bt\_status(status));

}

else

{

ALOGE("HAL REQUEST SUCCESS");

}

}

static void adapter\_state\_changed(bt\_state\_t state)

{

ALOGE("ADAPTER STATE UPDATED : %s", (state == BT\_STATE\_OFF)?"OFF":"ON");

if (state == BT\_STATE\_ON) {

bt\_enabled = 1;

} else {

bt\_enabled = 0;

}

}

static void devices\_found\_callback(int num\_properties, bt\_property\_t \*properties){

int i = 0;

pTmpBtInfo = (BT\_Info \*)malloc(sizeof(BT\_Info));

memset(pTmpBtInfo, 0x0, sizeof(BT\_Info));

for(i = 0; i < num\_properties; i++){

if (properties[i].type == BT\_PROPERTY\_BDADDR){

memcpy(pTmpBtInfo->bd\_addr, (char \*)properties[i].val, properties[i].len);

}

if(properties[i].type == BT\_PROPERTY\_BDNAME){

strcpy(pTmpBtInfo->bd\_name,(char \*)properties[i].val);

}

if(properties[i].type == BT\_PROPERTY\_REMOTE\_RSSI){

memcpy(&pTmpBtInfo->rssi,(int32\_t \*)properties[i].val,properties[i].len);

}

}

/\* insert into list \*/

if(g\_pBtListHear == NULL)

{

g\_pBtListHear = pTmpBtInfo;

}

else {

mTmpBtInfoForList = g\_pBtListHear;

while((mTmpBtInfoForList != NULL) && (mTmpBtInfoForList->pNext != NULL))

{

mTmpBtInfoForList = mTmpBtInfoForList->pNext;

}

mTmpBtInfoForList->pNext = pTmpBtInfo;

}

}

static void discovery\_state\_callback(bt\_discovery\_state\_t state) {

discovery\_state = (state == BT\_DISCOVERY\_STARTED) ? true : false;

if(discovery\_state == false)

Scanning\_state=false;

}

static bt\_callbacks\_t bt\_callbacks = {

sizeof(bt\_callbacks\_t),

adapter\_state\_changed,

NULL, /\*adapter\_properties\_cb \*/

NULL, /\* remote\_device\_properties\_cb \*/

devices\_found\_callback, /\* device\_found\_cb \*/

discovery\_state\_callback, /\* discovery\_state\_changed\_cb \*/

NULL, /\* pin\_request\_cb \*/

NULL, /\* ssp\_request\_cb \*/

NULL, /\*bond\_state\_changed\_cb \*/

NULL, /\* acl\_state\_changed\_cb \*/

NULL, /\* thread\_evt\_cb \*/

NULL, /\*dut\_mode\_recv\_cb \*/

NULL /\*le\_test\_mode\_cb\*/

};

void bt\_hal\_init(void)

{

ALOGE("INIT BT ");

status = (bt\_status\_t)sBtInterface->init(&bt\_callbacks);

check\_return\_status(status);

}

void bt\_enable(void)

{

ALOGE("ENABLE BT");

if (bt\_enabled) {

return;

}

status = (bt\_status\_t)sBtInterface->enable();

check\_return\_status(status);

}

void bt\_disable(void)

{

while(1){

if(bt\_init\_complete == 1)break;

usleep(100000);

}

ALOGE("DISABLE BT");

if (!bt\_enabled) {

return;

}

status = (bt\_status\_t)sBtInterface->disable();

check\_return\_status(status);

}

static bool startDiscovery(void) {

unsigned char result = false;

if (!sBtInterface) return result;

status = (bt\_status\_t)sBtInterface->start\_discovery();

result = (status == BT\_STATUS\_SUCCESS) ? true : false;

return result;

}

static void config\_permissions(void)

{

struct \_\_user\_cap\_header\_struct header;

struct \_\_user\_cap\_data\_struct cap;

header.pid = 0;

prctl(PR\_SET\_KEEPCAPS, 1, 0, 0, 0);

setuid(AID\_BLUETOOTH);

setgid(AID\_BLUETOOTH);

header.version = \_LINUX\_CAPABILITY\_VERSION;

cap.effective = cap.permitted = cap.inheritable =

1 << CAP\_NET\_RAW |

1 << CAP\_NET\_ADMIN |

1 << CAP\_NET\_BIND\_SERVICE |

1 << CAP\_SYS\_RAWIO |

1 << CAP\_SYS\_NICE |

1 << CAP\_SETGID;

capset(&header, &cap);

setgroups(sizeof(groups)/sizeof(groups[0]), groups);

}

bool is\_file(const char \*path)

{

struct stat statbuf;

if(lstat(path, &statbuf) ==0)

return S\_ISREG(statbuf.st\_mode) != 0;

return false;

}

static void \*BT\_Thread( void\* )

{

while(1){

if(bt\_enabled){

bt\_init\_complete =1;

startDiscovery();

break;

}

usleep(100000);

}

return 0;

}

void set\_scanning\_state(bool value){

Scanning\_state = value;

}

void updateTextInfo(BT\_Info \*pTmpBtInfoForList)

{

char cBuf[1024];

static int loop = 0;

int i = 0;

BT\_Info \* pInfoList = NULL;

mmi\_window \*window = new mmi\_window();

int w = window->get\_width();

int h = window->get\_height();

g\_module->add\_window(window);

int last\_text\_y = h / 10;

int last\_text\_x = w / 4;

mmi\_point\_t point;

point.x = last\_text\_x;

point.y = last\_text\_y;

pInfoList = pTmpBtInfoForList;

if(pTmpBtInfoForList == NULL){

if(Scanning\_state == true) {

if(loop == 0 ){

text\_waiting->set\_text("Status: Scanning -----");

loop = 1;

}else{

text\_waiting->set\_text("Status: Scanning +++++");

loop = 0;

}

}else{

text\_waiting->set\_text("----No dev found ----");

}

}else if(discovery\_state == true){

if(Scanning\_state == true){

if(loop == 0 ){

text\_waiting->set\_text("Status: Inquiring -----");

loop = 1;

}else{

text\_waiting->set\_text("Status: Inquiring +++++");

loop = 0;

}

}

}

else{

text\_waiting->set\_text("Status: Scanning Completed");

text[10]->set\_text("----End of Scan List----");

return;

}

while(pTmpBtInfoForList){

memset(cBuf,0x00,sizeof(cBuf));

if(strlen((const char\*)pTmpBtInfoForList->bd\_name)){

sprintf(cBuf + strlen(cBuf), " %.12s", (const char\*)pTmpBtInfoForList->bd\_name);

}else{

sprintf(cBuf, "%02x:%02x:%02x:%02x:%02x:%02x",

pTmpBtInfoForList->bd\_addr[0], pTmpBtInfoForList->bd\_addr[1], pTmpBtInfoForList->bd\_addr[2],

pTmpBtInfoForList->bd\_addr[3], pTmpBtInfoForList->bd\_addr[4], pTmpBtInfoForList->bd\_addr[5]);

}

pTmpBtInfoForList = pTmpBtInfoForList->pNext;

if(i<10){

text[i]->set\_text(cBuf);

i++;

}

}

return;

}

bool BT\_init(void){

config\_permissions();

if ( HAL\_load() < 0 ) {

}

if(is\_file(BT\_CONFIG\_OLD)){

remove(BT\_CONFIG\_OLD);

}

if(is\_file(BT\_CONFIG\_NEW)){

remove(BT\_CONFIG\_NEW);

}

bt\_hal\_init();

bt\_enable();

pthread\_create( &btThread, NULL, BT\_Thread, (void\*) NULL);

return true;

}

void BT\_deinit(void)

{

BT\_Info \*pTmpBtInfoForList = NULL;

bt\_disable();

while(1){

if((bt\_init\_complete == 1) && (bt\_enabled == 0) )break;

usleep(100000);

}

bt\_cleanup();

HAL\_unload();

if(is\_file(BT\_CONFIG\_OLD)){

remove(BT\_CONFIG\_OLD);

}

if(is\_file(BT\_CONFIG\_NEW)){

remove(BT\_CONFIG\_NEW);

}

while(g\_pBtListHear){

pTmpBtInfoForList = g\_pBtListHear;

g\_pBtListHear = g\_pBtListHear->pNext;

free(pTmpBtInfoForList);

}

pthread\_join(btThread, NULL);

bt\_init\_complete = -1;

return;

}

static void \*processThread(void \*)

{

signal(SIGUSR1, signalHandler);

mmi\_window \*window = new mmi\_window();

int w = window->get\_width();

int h = window->get\_height();

int i = 0;

g\_module->add\_window(window);

int last\_text\_y = h / 10;

int last\_text\_x = w / 4;

mmi\_point\_t point;

point.x = last\_text\_x;

point.y = last\_text\_y;

ALOGE("%s: Start\n", \_\_FUNCTION\_\_);

text\_waiting = new mmi\_text(point, "");

text\_waiting->set\_text("Status: Start");

g\_module->add\_text(text\_waiting);

for(i=0;i<11;i++){

last\_text\_y += 2 \* mmi\_text::get\_font\_size\_y();

point.y = last\_text\_y;

text[i] = new mmi\_text(point, "");

g\_module->add\_text(text[i]);

}

if(BT\_init() == false){

ALOGE( "%s: Exit\n", \_\_FUNCTION\_\_);

text\_waiting->set\_text("Status: INIT FAILED");

pthread\_exit(NULL);

return NULL;

}

bt\_init\_over = true;

if (exit\_thd)

goto exit;

text\_waiting->set\_text( "Status: Scanning");

set\_scanning\_state(true);

while (1) {

if (exit\_thd)

return NULL;

updateTextInfo(g\_pBtListHear);

usleep(200000);

}

exit:

ALOGE("%s: Exit\n", \_\_FUNCTION\_\_);

pthread\_exit(NULL);

return NULL;

}

void pass(void \*) {

if(!bt\_init\_over) return;

g\_retval = 0;

sem\_post(&g\_sem);

pthread\_kill(processThreadPid, SIGUSR1);

exit\_thd = true;

}

void fail(void \*) {

if(!bt\_init\_over) return;

g\_retval = -1;

pthread\_kill(processThreadPid, SIGUSR1);

exit\_thd = true;

sem\_post(&g\_sem);

}

void initUI() {

g\_module->add\_btn\_pass(pass);

g\_module->add\_btn\_fail(fail);

}

void get\_input() {

memset(&input, 0, sizeof(struct input\_params));

parse\_parameter(mmi\_config::query\_config\_value(g\_module->get\_domain(), "parameter"), paras);

}

void init() {

g\_retval = -1;

sem\_init(&g\_sem, 0, 0);

initUI();

}

void finish() {

sem\_wait(&g\_sem);

g\_module->clean\_source();

BT\_deinit();

}

void pcba\_test(){

int count = 0;

mmi\_window \*window = new mmi\_window();

mmi\_point\_t point = { window->get\_width() / 4, window->get\_height() / 3 };

ALOGE("%s: Start\n", \_\_FUNCTION\_\_);

text\_waiting = new mmi\_text(point, "Bluetooth Initializing...");

g\_module->add\_text(text\_waiting);

usleep(300000);

if(BT\_init() == false){

ALOGE( "%s: Exit\n", \_\_FUNCTION\_\_);

text\_waiting->set\_text("Bluetooth open Fail!");

g\_retval = -1;

usleep(500000);

}else{

while(count<30&&0==bt\_enabled){

count++;

usleep(100000);

}

if(bt\_enabled){

g\_retval = 0;

text\_waiting->set\_text("Bluetooth open OK!");

BT\_deinit();

usleep(500000);

}

}

g\_module->clean\_source();

}

int manual\_test() {

init();

int res = pthread\_create(&processThreadPid, NULL, processThread, NULL);

if(res < 0)

ALOGE("can't create pthread: %s\n", strerror(errno));

else

pthread\_join(processThreadPid, NULL);

finish();

return g\_retval;

}

int module\_main(mmi\_module \* mod) {

if(mod == NULL)

return -1;

g\_module = mod;

bt\_init\_over = false;

exit\_thd = false;

get\_input();

case\_run\_mode\_t mode = g\_module->get\_run\_mode();

if(mode == TEST\_MODE\_PCBA) {

pcba\_test();

return g\_retval;

} else {

return manual\_test();

}

}