|  |  |  |
| --- | --- | --- |
| УТВЕРЖДЕН |  |  |
| ВМАИЕПМ-ЛУ |  |  |
| УДОСТОВЕРЕН |  |  |
| ВМАИЕПМ-УЛ |  |  |

ЕМКА

Текст программы

ВМАИЕПМ

(ВМАИЕПМ-001ФЛ)

Листов 2

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Литера

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Инв. № подл. | Подп. и дата | Взам. инв. № | Инв. № дубл. | Подп. и дата |
|  |  |  |  |  |

# utils.cpp

#include "visa.h"  
#include "unmbase.h"  
#include "unmuem.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include "utils.h"  
#if defined(\_\_linux\_\_)  
#include "../../linux-wrappers/linux-wrappers.h"  
#endif  
  
#ifndef RETRIEVE\_ERROR\_MESSAGE  
#define RETRIEVE\_ERROR\_MESSAGE unmbase\_error\_message  
#endif  
  
void checkError(ViSession vi, ViStatus S, int flags)  
{  
 if (!(flags & ceSilent) || S != 0)  
 {  
 printf("\n Status returned: %08X\n", S);  
 }  
 if (S != 0)  
 {  
 char errstr[256];  
 ViStatus S1 = RETRIEVE\_ERROR\_MESSAGE(0, S, errstr);  
 if (!S1)  
 printf("\n Error string: %s\n", errstr);  
 else  
 printf("\n Error string: failed to retrieve (with status %08X)\n", S1);  
 if (flags & ceTracept)  
 printf("\n At trace point %d\n", unmuem\_dbg\_tracepoint\_q());  
 }  
 if ((flags & ceFail) && S < 0)  
 {  
 printf("\n This is a FATAL error\n");  
 exit(2);  
 }  
}  
  
char \*progname;  
  
void set\_progname(char \*av)  
{  
 char \*p = strrchr(av, '\\');  
 progname = p? p+1 : av;  
}  
  
static void usage\_\_()  
{  
 printf(  
 "Usage:\n"  
 " %s -s <slot number> [-m meznum] [-nidn]\n"  
 "or\n"  
 " %s -l <device logical address> [-m meznum] [-nidn]\n"  
 "or\n"  
 " %s -r <resource name string> [-m meznum] [-nidn]\n"  
 "or\n"  
 " %s -lst\n",  
 progname, progname, progname, progname  
 );  
}  
  
  
//typedef void (\* usage\_f)();  
static usage\_f usage = usage\_\_;  
  
void set\_usage(usage\_f f) { usage = f; }  
  
static int verbose = 1;  
  
void set\_verbose(int v)  
{  
 verbose = v;  
}  
  
  
int selectDevice(int argc, char \*argv[], char \*idstr, bool \*doIdn, int \*meznum)  
{  
 set\_progname(argv[0]);  
#if defined \_\_linux\_\_  
 setbuf(stdout, NULL);  
#endif  
 printf("Called: %s", progname);  
 for (int i = 1; i < argc; i++) printf(" %s", argv[i]);  
 printf("\n");  
  
 if (argc < 2)  
 {  
 printf("Device to open not defined."); usage();  
 return -1;  
 }  
  
 enum addrMode {Slot, LA, RSId, List} aMode;  
  
 if (strcmp(argv[1], "-l") == 0)  
 aMode = LA;  
 else if (strcmp(argv[1], "-s") == 0)  
 aMode = Slot;  
 else if (strcmp(argv[1], "-r") == 0)  
 aMode = RSId;  
 else if (strcmp(argv[1], "-lst") == 0)  
 aMode = List;  
 else  
 {  
 usage();  
 return -1;  
 }  
  
 int addr = 0;  
 bool selected = false;  
 \*doIdn = true;  
 int proceedarg;  
 \*meznum = -1;  
  
 if (aMode != List)  
 {  
 if (argc < 3)  
 {  
 printf("Device to open not defined."); usage();  
 return -1;  
 }  
 }  
  
 if (aMode == Slot || aMode == LA)  
 {  
 addr = atoi(argv[2]);  
 if (!addr)  
 {  
 usage();  
 return -1;  
 }  
 }  
 else if (aMode == RSId)  
 {  
 strcpy(idstr, argv[2]);  
 selected = true;  
 }  
  
 proceedarg = 2;  
  
 if (argc >= 4)  
 {  
 if (strcmp(argv[3], "-m") == 0)  
 {  
 if (argc <= 4)  
 {  
 usage();  
 return 1;  
 }  
 \*meznum = atoi(argv[4]);  
 proceedarg += 2;  
 }  
 }  
  
 if (argc >= proceedarg + 2)  
 {  
 if (strcmp(argv[proceedarg + 1], "-nidn") == 0)  
 \*doIdn = false;  
 }  
  
 ViStatus S;  
 int plusFlag = verbose? 0 : ceSilent;  
  
 if (aMode == Slot || aMode == LA || aMode == List)  
 {  
 printf("\nSearching for device...\n\n");  
  
 ViSession RM, vi;  
 ViFindList list;  
 ViUInt32 n;  
 char devname[256];  
  
 printf("viOpenDefaultRM\n");  
 S = viOpenDefaultRM(&RM);  
 checkError(0, S, ceFail+plusFlag);  
  
 printf("viFindRsrc()\n");  
 //S = viFindRsrc(RM, "VXI?\*::INSTR", &list, &n, devname);  
 //S = viFindRsrc(RM, "VXI?\*", &list, &n, devname);  
 S = viFindRsrc(RM, (ViString)"?\*", &list, &n, devname);  
 checkError(0, S, ceFail+plusFlag);  
  
 while (1)  
 {  
 printf("viOpen()\n");  
 S = viOpen(RM, devname, VI\_NO\_LOCK, 0, &vi);  
 checkError(0, S, plusFlag);  
 if (S >= 0)  
 {  
 ViUInt16 vxi\_la = 0;  
 ViUInt16 manf\_id = 0;  
 ViUInt16 model\_code = 0;  
 ViUInt16 crate = 0;  
 ViUInt16 slot = 0;  
  
 if (verbose || aMode == List)   
 printf("Getting device attrs...\n");  
 ViStatus S1,S2 = 0;  
 S1 = viGetAttribute(vi, VI\_ATTR\_VXI\_LA, &vxi\_la);  
 if (!S2 && S1) S2 = S1;  
 S2 = viGetAttribute(vi, VI\_ATTR\_MANF\_ID, &manf\_id);  
 if (!S2 && S1) S2 = S1;  
 S1 = viGetAttribute(vi, VI\_ATTR\_MODEL\_CODE, &model\_code);  
 if (!S2 && S1) S2 = S1;  
 S1 = viGetAttribute(vi, VI\_ATTR\_MAINFRAME\_LA, &crate);  
 if (!S2 && S1) S2 = S1;  
 S1 = viGetAttribute(vi, VI\_ATTR\_SLOT, &slot);  
 if (!S2 && S1) S2 = S1;  
 if (S2)  
 {  
 printf("Failed to retrieve some attrs\n");  
 checkError(vi, S2, 0);  
 }  
  
 if (verbose || aMode == List)  
 printf(  
 "ID: %s\n"  
 "LA: %04X\n"  
 "CRATE: %04X\n"  
 "SLOT: %04X\n"  
 "MAINF\_ID: %04X\n"  
 "MODEL\_CODE: %04X\n",  
 devname, vxi\_la, crate, slot, manf\_id, model\_code  
 );  
  
 if (aMode == Slot && slot == addr ||  
 aMode == LA && vxi\_la == addr)  
 {  
 if (verbose || aMode == List)  
 printf("THIS DEVICE IS SELECTED\n");  
 selected = true;  
 strcpy(idstr, devname);  
 }  
 printf("viClose()\n");  
 S = viClose(vi);  
 checkError(0, S, plusFlag);  
 }  
  
 if (!--n) break;  
  
 printf("\nviFindNext()\n");  
 S = viFindNext(list, devname);  
 checkError(0, S, ceFail+plusFlag);  
 printf("\n");  
 }  
  
 printf("\nviClose(search)\n");  
 S = viClose(list);  
 checkError(0, S, ceFail+plusFlag);  
 printf("viClose(RM)\n");  
 S = viClose(RM);  
 checkError(0, S, ceFail+plusFlag);  
 }  
  
 if (aMode == List)  
 return 0;  
  
 if (!selected)  
 {  
 printf("\nDEVICE NOT FOUND\n\n");  
 return -3;  
 }  
  
 return 1;  
}  
  
ViStatus find\_mezonine(ViSession vi, int \*meznum)  
{  
 ViInt16 present, mtype;  
 ViStatus S;  
 for (int mn = 1; mn <= 4; mn++)  
 {  
 S = unmbase\_m\_type\_q(vi, mn, &present, &mtype);  
 if (S < 0)  
 {  
 \*meznum = 1;  
 return S;  
 }  
 if (!present) continue;  
 if (((mtype ^ UNMUEM\_MODEL\_CODE) & 0x00FF) == 0)  
 {  
 \*meznum = mn;  
 return S;  
 }  
 }  
 \*meznum = 1;  
 return S;  
}

# stdafx.cpp

// stdafx.cpp : source file that includes just the standard includes  
// AccessTest.pch will be the pre-compiled header  
// stdafx.obj will contain the pre-compiled type information  
  
#include "stdafx.h"  
  
// TODO: reference any additional headers you need in STDAFX.H  
// and not in this file

# vq.cpp

//  
#include "visa.h"  
#include "unmbase.h"  
#include "unmuem.h"  
#include "unmuem\_struct.h"  
#include "utils.h"  
#include <stdio.h>  
#include <conio.h>  
#include <string.h>  
#include <windows.h>  
#include <locale.h>  
  
//extern char \*progname;  
  
int testRead();  
int testWrite();  
void testInteractions1();  
void backLine();  
  
static ViUInt32 memsize;  
  
enum testid\_t {tiUndef, tiRead, tiWrite};  
  
int main(int argc, char\* argv[])  
{  
 ViStatus S;  
 static char idstr[256];  
 bool doIdn = true;  
 int meznum = -1;  
  
 setlocale(LC\_ALL, "Russian");  
 set\_progname(argv[0]);  
  
 int sResult = selectDevice(argc, argv, idstr, &doIdn, &meznum);  
  
 if (sResult <= 0)  
 return -sResult;  
  
 ViSession vi/\*, mvi, ivi\*/;  
 printf("\nOpening the device with ID string %s...\n\n", idstr);  
  
 printf("unmbase\_init()\n");  
 S = unmbase\_init (idstr, VI\_ON, VI\_ON, &vi);  
 checkError(0, S);  
  
 printf("\nDevice opened successfully\n\n");  
  
 ViChar vdrv[256], vinstr[256];  
 printf("unmbase\_revision\_query()\n");  
 S = unmbase\_revision\_query (vi, vdrv, vinstr);  
 checkError(0, S);  
  
 printf("\nDRV version = %s, INSTR version = %s\n\n", vdrv, vinstr);  
  
  
 printf("\nClosing the device...\n\n");  
  
 printf("unmbase\_close()\n");  
 S = unmbase\_close(vi);  
 checkError(vi, S);  
  
 printf("\nDevice closed successfully\n\n");  
  
 return 0;  
}

# HParser.cpp

#include "HParser.h"  
  
  
ViUInt16 ChParsingMT(ViSession mvi, )  
{  
 ViUInt16 retVal = 0;  
  
  
 return retVal;  
}

# octest.cpp

#include "visa.h"  
#include "unmbase.h"  
#include "unmuem.h"  
//#include "unmuem\_struct.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <windows.h>  
#include <locale.h>  
#include "utils.h"  
//using namespace std;  
  
int test16bitRegs(ViSession mvi);  
int testLED(ViSession mvi);  
  
int main(int argc, char\* argv[])  
{  
 ViSession S;  
 char idstr[256];  
 bool doIdn = true;  
 int meznum = -1;  
  
 setlocale(LC\_ALL, "Russian");  
  
 int sResult = selectDevice(argc, argv, idstr, &doIdn, &meznum);  
  
 if (sResult <= 0)  
 return -sResult;  
  
 printf("\nOpening the device with ID string %s...\n\n", idstr);  
  
 ViSession vi, mvi;  
  
 printf("unmbase\_init()\n");  
 S = unmbase\_init (idstr, VI\_ON, VI\_ON, &vi);  
 checkError(0, S);  
  
 if (meznum < 1)  
 {  
 printf("find\_mezonine()\n");  
 S = find\_mezonine(vi, &meznum);  
 checkError(vi, S);  
 }  
  
 printf("unmuem\_init()\n");  
 S = unmuem\_init(idstr,VI\_ON, VI\_ON, &mvi);  
 checkError(0, S);  
  
 printf("unmuem\_connect()\n");  
 S = unmuem\_connect(mvi,vi,1,doIdn, VI\_ON);  
 checkError(mvi, S);  
  
 printf("\nDevice opened successfully\n\n");  
  
 /\*  
 ViSession ivi;  
 printf("unmuem\_dbg\_get\_intssn()\n");  
 S = unmuem\_dbg\_get\_intssn(mvi, &ivi);  
 checkError(mvi, S);  
 \*/  
  
 //test16bitRegs(ivi);  
  
 test16bitRegs(mvi);  
 //!!! This one works as well as test16bitRegs(ivi) !!!  
  
 testLED(mvi);  
  
 printf("\nClosing the device...\n\n");  
  
 printf("unmuem\_close()\n");  
 S = unmuem\_close(mvi);  
 checkError(mvi, S);  
  
 printf("unmbase\_close()\n");  
 S = unmbase\_close(vi);  
 checkError(mvi, S);  
  
 printf("\nDevice closed successfully\n\n");  
  
 return 0;  
}  
  
#define A(reg) (reg << 1)  
  
void read16reg(ViSession mvi, ViUInt32 addr)  
{  
 ViStatus S;  
 ViUInt16 Data;  
 printf("unmbase\_m\_in16 (REG: %02X (ADDR: %02X))\n", addr, A(addr));  
 S = unmbase\_m\_in16 (mvi, A(addr), &Data);  
 checkError(mvi, S, ceFail);  
 printf(" READ: %04X\n", Data);  
}  
  
int rw16regVal(ViSession mvi, ViUInt32 addr, ViUInt16 value)  
{  
 ViStatus S;  
 ViUInt16 Data;  
 printf("unmbase\_m\_out16 (REG: %02X (ADDR: %02X), WRITE: %04X)\n", addr, A(addr), value);  
 S = unmbase\_m\_out16 (mvi, A(addr), value);  
 checkError(mvi, S, ceFail);  
 printf("unmbase\_m\_in16 (REG: %02X (ADDR: %02X))\n", addr, A(addr));  
 S = unmbase\_m\_in16 (mvi, A(addr), &Data);  
 checkError(mvi, S, ceFail);  
 bool match = Data == value;  
 printf(" READ: %04X (%s)\n", Data, match? "MATCH" : "MISMATCH");  
 return match? 0 : 1;  
}  
  
int rw16reg(ViSession mvi, ViUInt32 addr)  
{  
 int result = 0;  
 result |= rw16regVal(mvi, addr, 0x0000);  
 result |= rw16regVal(mvi, addr, 0x5555);  
 result |= rw16regVal(mvi, addr, 0xAAAA);  
 result |= rw16regVal(mvi, addr, 0xFFFF);  
 result |= rw16regVal(mvi, addr, 0x0000);  
 return result;  
}  
  
int test16bitRegs(ViSession mvi)  
{  
 printf(  
 "\nThis test will perform reading from low half of all registers\n"  
 "and read/write test in low half of ACNTRL 1/2 registes.\n"  
 "It uses direct UNMBASE i/o functions instead of UNMUEM i/o functions\n");  
  
 ViStatus S;  
 ViUInt16 Data;  
  
 printf("unmbase\_m\_in16 (UNMUEM\_MEZ\_ID (REG: %02X, ADDR: %02X))\n", UNMUEM\_MEZ\_ID, A(UNMUEM\_MEZ\_ID));  
 S = unmbase\_m\_in16 (mvi, A(UNMUEM\_MEZ\_ID), &Data);  
 checkError(mvi, S, ceFail);  
 printf(" READ: %04X\n\n", Data);  
  
 for (ViUInt32 reg = 0; reg < 16; reg++)  
 read16reg(mvi, reg);  
  
 printf("\n");  
  
 int result = 0;  
 result |= rw16reg(mvi, UNMUEM\_ACNTRL\_1);  
 printf("\n");  
 result |= rw16reg(mvi, UNMUEM\_ACNTRL\_2);  
  
 printf("\nRegister tests %s\n", result? "FAILED" : "SUCCEED");  
  
 return result;  
}  
  
void LEDswitch(ViSession mvi, bool on)  
{  
 ViStatus S;  
 ViUInt16 value;  
 ViUInt32 addr = UNMUEM\_MEZ\_CMD;  
 value = on? 0x0100 : 0;  
 printf("unmbase\_m\_out16 (MEZ\_CMD (REG: %02X (ADDR: %02X)), WRITE: %04X)\n", addr, A(addr), value);  
 S = unmbase\_m\_out16 (mvi, A(addr), value);  
 checkError(mvi, S, ceFail);  
}  
  
int testLED(ViSession mvi)  
{  
 printf("\nLED test in progress. See internal LED!\n");  
 for (int i = 0; i < 5; i++)  
 {  
 LEDswitch(mvi, true);  
 Sleep(1000);  
 LEDswitch(mvi, false);  
 Sleep(1000);  
 }  
 return 0;  
}

# linux-wrappers-test.cpp

#include <stdio.h>  
#include <stdlib.h>  
#ifdef WIN32  
#include <Windows.h>  
#else  
#include <pthread.h>  
#include <unistd.h>  
#endif  
#include "linux-wrappers.h"  
  
#define MAGIC 555  
  
#if defined(\_\_linux\_\_)  
#define LPDWORD uint32\_t\*  
#define PASSED "\033[01;32mPASSED\033[0m"  
#define FAILED "\033[01;31mFAILED\033[0m"  
#else  
#undef FAILED  
#define PASSED "PASSED"  
#define FAILED "FAILED"  
#define usleep(x) Sleep(x/1000)  
#endif  
  
#if defined(\_\_linux\_\_)  
void\*  
#else  
DWORD WINAPI  
#endif  
thread\_fn(void \*arg)  
{  
 int i;  
 long int ret = 0;  
  
 for (i = 0; i < 5; i++) {  
 printf("%s-%d\n", (char \*)arg, i);  
  
 usleep(500000);  
 }  
 ret = MAGIC;  
#if defined(\_\_linux\_\_)  
 pthread\_exit((void \*)ret);  
#else  
 ExitThread(ret);  
#endif  
}  
  
#if defined(\_\_linux\_\_)  
void show\_priority(struct sched\_param param, int policy)  
{  
 printf("\tsheduling = %s\n\tpriority = %d\n",  
 policy == SCHED\_FIFO ? "SCHED\_FIFO" :  
 policy == SCHED\_RR ? "SCHED\_RR" :  
 policy == SCHED\_OTHER ? "SCHED\_OTHER" : "N/A", param.sched\_priority);  
}  
#endif  
  
int main()  
{  
#if defined(\_\_linux\_\_)  
 struct sched\_param param;  
 HANDLE self = 0;  
#endif  
 HANDLE thread = 0;  
 int excode;  
 const char \*test\_str = "thread\_test\_str";  
 int policy, new\_policy;  
 int ret = 1, ec\_ret = 1;  
  
 thread = CreateThread(NULL, NULL, thread\_fn, const\_cast<char \*>(test\_str), 0, NULL);  
 Sleep(10);  
 printf("CreateThread test %s\n", thread ? PASSED : FAILED);  
 if (!thread)  
 return 1;  
 ret = 1;  
#if defined (\_\_linux\_\_)  
 printf("priority:\n");  
 pthread\_getschedparam(thread, &policy, &param);  
 new\_policy = policy;  
 show\_priority(param, policy);  
 printf("new priority:\n");  
#endif  
 if (ret &= SetThreadPriority(thread, THREAD\_PRIORITY\_TIME\_CRITICAL)) {  
#if defined (\_\_linux\_\_)  
 pthread\_getschedparam(thread, &new\_policy, &param);  
 show\_priority(param, new\_policy);  
#endif  
 } else {  
 printf("error: priority is NOT set.\n");  
 }  
#if defined (\_\_linux\_\_)  
 printf("SetThreadPriority test %s\n", (policy != new\_policy || policy == SCHED\_FIFO  
 || policy == SCHED\_RR) ? PASSED : FAILED);  
#endif  
 ec\_ret &= GetExitCodeThread(thread, (LPDWORD)&excode);  
 printf("excode = %s (%d), ret = %d\n", excode == STILL\_ACTIVE ? "STILL\_ACTIVE" : "unknown", excode, ret);  
#if defined (\_\_linux\_\_)  
 pthread\_join(thread, (void \*\*)&excode);  
#else  
 WaitForSingleObject(thread, INFINITE);  
 ec\_ret &= GetExitCodeThread(thread, (LPDWORD)&excode);  
#endif  
 printf("excode = %s (%d), ret = %d\n", excode == STILL\_ACTIVE ? "STILL\_ACTIVE" : "unknown", excode, ret);  
 ret &= ec\_ret;  
 printf("GetExitCodeThread exit code test %s\n", ((bool)ec\_ret == true) && (excode == MAGIC) ? PASSED : FAILED);  
#if defined(\_\_linux\_\_)  
 ret = !ret;  
#endif  
  
 return ret;  
}

# sample.cpp

#include "visa.h"  
#include "unmbase.h"  
#include "unmuem.h"  
#include "unmuem\_struct.h"  
#include <stdio.h>  
#include <stdlib.h>//for atoi  
#include <iostream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
 printf("Hello, world!\n");  
 cout << "Hello, world! (2)\n";  
 cout << "This is a "<<(unmuem\_drv\_caps() & 1? "sumulator" : "real")<<" driver.\n";  
  
 const char   
 \*name = "VXI1::1::INSTR",  
 \*snum = "1";  
 int num;  
  
 if (argc >= 2)  
 name = argv[1];  
 if (argc >= 3)  
 snum = argv[2];  
 num = atoi(snum);  
  
 cout << "Connecting to NM "<<name<<" mez N "<<num<<"\n";  
  
 ViStatus S;  
 ViSession vi, mvi;  
  
 S = unmbase\_init (const\_cast<char \*>(name), VI\_ON, VI\_ON, &vi);  
 if (S) { cout << "Fail "<<S<<" unmbase\_init()\n"; return 1; }  
 S = unmuem\_init(const\_cast<char \*>(name),VI\_ON, VI\_ON, &mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_init()\n"; return 1; }  
 S = unmuem\_connect(mvi,vi,num,VI\_ON, VI\_ON);  
 if (S) { cout << "Fail "<<S<<" unmuem\_connect()\n"; return 1; }  
  
 cout << "Device has been opened\n";  
  
 char rev1[256], rev2[256];  
 S = unmuem\_revision\_query(mvi, rev1, rev2);  
 if (S) { cout << "Fail "<<S<<" unmuem\_revision\_query()\n"; return 1; }  
 cout << "Revisions: "<<rev1<<" "<<rev2<<"\n";  
  
 ViUInt32 testv = 0x0505A0A0, backv;  
 S = unmuem\_reg\_write(mvi, UNMUEM\_ACNTRL\_1, testv);  
 if (S) { cout << "Fail "<<S<<" unmuem\_reg\_write()\n"; return 1; }  
 S = unmuem\_reg\_read(mvi, UNMUEM\_ACNTRL\_1, &backv);  
 if (S) { cout << "Fail "<<S<<" unmuem\_reg\_read()\n"; return 1; }  
 if (backv == testv)  
 cout <<"Reg read/write test passed.\n";  
 else  
 printf("Reg read/write test FAILED: written %08X, read %08X.\n", testv, backv);  
  
 ViUInt32 nw;  
 testv = 0x0505A0A1;  
 S = unmuem\_block\_write(mvi, UNMUEM\_RAM\_START, &testv, 1, &nw);  
 if (S) { cout << "Fail "<<S<<" unmuem\_block\_write()\n"; return 1; }  
  
 S = unmuem\_fifo\_push(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_fifo\_push()\n"; return 1; }  
  
 S = unmuem\_dbg\_block\_read(mvi, UNMUEM\_RAM\_START, &backv, 1);  
 if (S) { cout << "Fail "<<S<<" unmuem\_dbg\_block\_read()\n"; return 1; }  
  
 cout <<"FIFO in test "<<(testv == backv?"passed":"FAILED")<<".\n";  
  
 testv = 0x0505A0A2;  
 S = unmuem\_reg\_write(mvi, UNMUEM\_ACNTRL\_1, testv);  
 if (S) { cout << "Fail "<<S<<" unmuem\_reg\_write()\n"; return 1; }  
  
 /\*  
 for (int i = 0; i < 32; i++)  
 {  
 S = unmuem\_send\_reg (mvi, UNMUEM\_ACNTRL\_1);  
 if (S) { cout << "Fail "<<S<<" unmuem\_send\_reg()\n"; return 1; }  
 }  
 \*/  
  
 unmuem\_Frame\_t f;  
 S = unmuem\_fifo\_read(mvi, (ViUInt32\*)&f, 1, &nw);  
 if (S) { cout << "Fail "<<S<<" unmuem\_fifo\_read()\n"; return 1; }  
 if (nw != 1) cout << "FIFO out test failed - no words\n";  
 else if (f.head.b.address != UNMUEM\_ACNTRL\_1 || f.data != testv)  
 cout << "FIFO out test failed - data mismatch\n";  
 else  
 cout << "FIFO out test passed\n";  
  
 S = unmuem\_sem\_lock(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_sem\_lock()\n"; return 1; }  
  
 S = unmuem\_sem\_lock(mvi);  
 if (S == VI\_SUCCESS) cout << "unmuem\_sem\_lock() double success - this is an error\n";  
 else if (S != UNMUEM\_WARN\_SEM\_REJECT) { cout << "Fail "<<S<<" unmuem\_sem\_lock()\n"; return 1; }  
 else {  
 S = unmuem\_sem\_unlock(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_sem\_unlock()\n"; return 1; }  
 S = unmuem\_sem\_lock(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_sem\_lock()\n"; return 1; }  
 S = unmuem\_sem\_unlock(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_sem\_unlock()\n"; return 1; }  
 cout << "Semaphore test passed\n";  
 }  
  
  
 S = unmuem\_close(mvi);  
 if (S) { cout << "Fail "<<S<<" unmuem\_close()\n"; }  
 S = unmbase\_close(vi);  
 if (S) { cout << "Fail "<<S<<" unmbase\_close()\n"; }  
  
 printf("Bye, world!\n");  
  
 return 0;  
}

# main.cpp

#include <visatype.h>  
#include "unmbase.h"  
#include "unmuem.h"  
#include "fakedriver.h"  
#include "utils.h"  
#include <stdio.h>  
  
#include <locale.h>  
  
static const int MB = 1024\*1024;  
  
ViStatus testalloc(ViSession fkd, ViUInt32 memsize)  
 // memsize in megabytes  
{  
 ViStatus S;  
 printf("fkd\_alloc(, 0, %d MB)\n", memsize);  
 S = fkd\_alloc(fkd, 0, memsize \* MB);  
 checkErrorS(fkd, S);  
 if (S < 0) return S;  
 printf("fkd\_alloc(, 1, %d MB)\n", memsize);  
 S = fkd\_alloc(fkd, 1, memsize \* MB);  
 checkErrorS(fkd, S);  
 return S;  
}  
  
int main(int argc, char\* argv[])  
{  
  
 ViStatus S;  
 ViSession vi, mvi, fkd;  
 static char idstr[256];  
 bool doIdn = true;  
 int meznum = -1;  
 ViUInt32 memsize = 0;  
  
 set\_progname(argv[0]);  
 set\_verbose(0);  
  
 setlocale(LC\_ALL, "Russian");  
  
 int sResult = selectDevice(argc, argv, idstr, &doIdn, &meznum);  
  
 if (sResult <= 0)  
 return -sResult;  
  
 printf("unmbase\_init()\n");  
 S = unmbase\_init (idstr, VI\_ON, VI\_ON, &vi);  
 checkErrorS(0, S);  
  
 printf("unmbase\_mem\_q()\n");  
 S = unmbase\_mem\_q(vi, &memsize);  
 checkErrorS(0, S);  
  
 printf("TOTAL MEMORY SIZE = %d MB\n", memsize / MB);  
  
 printf("fkd\_init()\n");  
 S = fkd\_init(idstr,VI\_ON, VI\_ON, &fkd);  
 checkErrorS(0, S);  
  
 printf("fkd\_connect()\n");  
 S = fkd\_connect(fkd,vi, 2 ,VI\_ON, VI\_ON);  
 checkErrorS(fkd, S);  
  
 //testalloc(fkd, 60);  
   
 printf("unmuem\_init()\n");  
 S = unmuem\_init(idstr,VI\_ON, VI\_ON, &mvi);  
 checkErrorS(0, S);  
  
 if (meznum < 1)  
 {  
 printf("find\_mezonine()\n");  
 S = find\_mezonine(vi, &meznum);  
 checkErrorS(vi, S);  
 }  
  
 printf("unmuem\_connect()\n");  
 S = unmuem\_connect(mvi,vi,meznum,doIdn, VI\_ON);  
 checkErrorS(mvi, S);  
  
 //testalloc(fkd, 16);  
  
 printf("unmuem\_close()\n");  
 S = unmuem\_close(mvi);  
 checkErrorS(mvi, S);  
  
 printf("fkd\_close()\n");  
 S = fkd\_close(fkd);  
 checkErrorS(fkd, S);  
  
 printf("unmbase\_close()\n");  
 S = unmbase\_close(vi);  
 checkErrorS(vi, S);  
  
 return 0;  
}

| Лист регистрации изменений | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего  листов  (страниц)  в документе | Номер  документа | Входящий  номер сопрово-  дительного  документа  и дата | Под- пись | Дата |
| изме-  ненных | заменен-  ных | новых | аннули-  рован-  ных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |