```
  1: /*
  2:   libxbee - a C library to aid the use of Digi's Series 1 XBee modules
  3:             running in API mode (AP=2).
  4:
  5:   Copyright (C) 2009  Attie Grande (attie@attie.co.uk)
  6:
  7:   This program is free software: you can redistribute it and/or modify
  8:   it under the terms of the GNU General Public License as published by
  9:   the Free Software Foundation, either version 3 of the License, or
 10:   (at your option) any later version.
 11:
 12:   This program is distributed in the hope that it will be useful,
 13:   but WITHOUT ANY WARRANTY; without even the implied warranty of
 14:   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 15:   GNU General Public License for more details.
 16:
 17:   You should have received a copy of the GNU General Public License
 18:   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 19: */
 20:
 21: /* ######################################################################## */
 22: /* ### Win32 Code ######################################################## */
 23: /* ######################################################################## */
 24:
 25: /*  this file contains code that is used by Win32 ONLY */
 26: #ifndef _WIN32
 27: #error "This file should only be used on a Win32 system"
 28: #endif
 29:
 30: #include "win32.h"
 31: #include "win32.dll.c"
 32:
 33: static int init_serial(xbee_hnd xbee, int baudrate) {
 34:   int chosenbaud;
 35:   DCB tc;
 36:   int evtMask;
 37:   COMMTIMEOUTS timeouts;
 38:
 39:   /* open the serial port */
 40:   xbee->tty = CreateFile(TEXT(xbee->path),
 41:                          GENERIC_READ | GENERIC_WRITE,
 42:                          0,    /* exclusive access */
 43:                          NULL, /* default security attributes */
 44:                          OPEN_EXISTING,
 45:                          FILE_FLAG_OVERLAPPED,
 46:                          NULL);
 47:   if (xbee->tty == INVALID_HANDLE_VALUE) {
 48:     xbee_logS("Invalid file handle...");
 49:     xbee_logE("Is the XBee plugged in and avaliable on the correct port?");
 50:     xbee_mutex_destroy(xbee->conmutex);
 51:     xbee_mutex_destroy(xbee->pktmutex);
 52:     xbee_mutex_destroy(xbee->sendmutex);
 53:     Xfree(xbee->path);
 54:     return -1;
 55:   }
 56:
 57:   GetCommState(xbee->tty, &tc);
 58:   tc.BaudRate          = baudrate;
 59:   tc.fBinary           = TRUE;
 60:   tc.fParity           = FALSE;
 61:   tc.fOutxCtsFlow      = FALSE;
 62:   tc.fOutxDsrFlow      = FALSE;
 63:   tc.fDtrControl       = DTR_CONTROL_DISABLE;
 64:   tc.fDsrSensitivity   = FALSE;
 65:   tc.fTXContinueOnXoff = FALSE;
 66:   tc.fOutX             = FALSE;
 67:   tc.fInX              = FALSE;
 68:   tc.fErrorChar        = FALSE;
 69:   tc.fNull             = FALSE;
 70:   tc.fRtsControl       = RTS_CONTROL_DISABLE;
 71:   tc.fAbortOnError     = FALSE;
 72:   tc.ByteSize          = 8;
 73:   tc.Parity            = NOPARITY;
 74:   tc.StopBits          = ONESTOPBIT;
 75:   SetCommState(xbee->tty, &tc);
 76:
 77:   timeouts.ReadIntervalTimeout = MAXDWORD;
 78:   timeouts.ReadTotalTimeoutMultiplier = 0;
 79:   timeouts.ReadTotalTimeoutConstant = 0;
 80:   timeouts.WriteTotalTimeoutMultiplier = 0;
 81:   timeouts.WriteTotalTimeoutConstant = 0;
 82:   SetCommTimeouts(xbee->tty, &timeouts);
 83:
 84:   SetCommMask(xbee->tty, EV_RXCHAR);
 85:
```

```
  86:   return 0;
  87: }
  88:
  89: /* a replacement for the linux select() function... for a serial port */
  90: static int xbee_select(xbee_hnd xbee, struct timeval *timeout) {
  91:   int evtMask = 0;
  92:   COMSTAT status;
  93:   int ret;
  94:
  95:   for (;;) {
  96:     /* find out how many bytes are in the Rx buffer... */
  97:     if (ClearCommError(xbee->tty,NULL,&status) && (status.cbInQue > 0)) {
  98:       /* if there is data... return! */
  99:       return 1; /*status.cbInQue;*/
 100:     } else if (timeout && timeout->tv_sec == 0 && timeout->tv_usec == 0) {
 101:       /* if the timeout was 0 (return immediately) then return! */
 102:       return 0;
 103:     }
 104:
 105:     /* otherwise wait for an Rx event... */
 106:     memset(&(xbee->ttyovrs),0,sizeof(OVERLAPPED));
 107:     xbee->ttyovrs.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
 108:     if (!WaitCommEvent(xbee->tty,&evtMask,&(xbee->ttyovrs))) {
 109:       if (GetLastError() == ERROR_IO_PENDING) {
 110:         DWORD timeoutval;
 111:         if (!timeout) {
 112:           /* behave like the linux function... if the timeout pointer was NULL
 113:              then wait indefinately */
 114:           timeoutval = INFINITE;
 115:         } else {
 116:           /* Win32 doesn't give the luxury of microseconds and seconds... just miliseconds! */
 117:           timeoutval = (timeout->tv_sec * 1000) + (timeout->tv_usec / 1000);
 118:         }
 119:         ret = WaitForSingleObject(xbee->ttyovrs.hEvent,timeoutval);
 120:         if (ret == WAIT_TIMEOUT) {
 121:           /* cause the WaitCommEvent() call to stop */
 122:           SetCommMask(xbee->tty, EV_RXCHAR);
 123:           /* if a timeout occured, then return 0 */
 124:           CloseHandle(xbee->ttyovrs.hEvent);
 125:           return 0;
 126:         }
 127:       } else {
 128:         return -1;
 129:       }
 130:     }
 131:     CloseHandle(xbee->ttyovrs.hEvent);
 132:   }
 133:
 134:   /* always return -1 (error) for now... */
 135:   return -1;
 136: }
 137:
 138: /* this offers the same behavior as non-blocking I/O under linux */
 139: int xbee_write(xbee_hnd xbee, const void *ptr, size_t size) {
 140:   xbee->ttyeof = FALSE;
 141:   if (!WriteFile(xbee->tty, ptr, size, NULL, &(xbee->ttyovrw)) &&
 142:       (GetLastError() != ERROR_IO_PENDING)) return 0;
 143:   if (!GetOverlappedResult(xbee->tty, &(xbee->ttyovrw), &(xbee->ttyw), TRUE)) {
 144:     if (GetLastError() == ERROR_HANDLE_EOF) xbee->ttyeof = TRUE;
 145:     return 0;
 146:   }
 147:   return xbee->ttyw;
 148: }
 149:
 150: /* this offers the same behavior as non-blocking I/O under linux */
 151: int xbee_read(xbee_hnd xbee, void *ptr, size_t size) {
 152:   xbee->ttyeof = FALSE;
 153:   if (!ReadFile(xbee->tty, ptr, size, NULL, &(xbee->ttyovrr)) &&
 154:       (GetLastError() != ERROR_IO_PENDING)) return 0;
 155:   if (!GetOverlappedResult(xbee->tty, &(xbee->ttyovrr), &(xbee->ttyr), TRUE)) {
 156:     if (GetLastError() == ERROR_HANDLE_EOF) xbee->ttyeof = TRUE;
 157:     return 0;
 158:   }
 159:   return xbee->ttyr;
 160: }
 161:
 162: /* this is because Win32 has some weird memory management rules...
 163:    - the thread that allocated the memory, MUST free it... */
 164: void xbee_free(void *ptr) {
 165:   if (!ptr) return;
 166:   free(ptr);
 167: }
 168:
 169: /* win32 equivalent of unix gettimeofday() */
 170: int gettimeofday(struct timeval *tv, struct timezone *tz) {
```

```
171:   if (tv) {
172:     struct _timeb timeb;
173:     _ftime(&timeb);
174:     tv->tv_sec = timeb.time;
175:     tv->tv_usec = timeb.millitm * 1000;
176:   }
177:   /* ignore tz for now */
178:   return 0;
179: }
180:
181: /* ###################################################################### */
182: /* ### Helper Functions (Mainly for VB6 use) ############################ */
183: /* ###################################################################### */
184:
185: /* enable the debug output to a custom file or fallback to stderr */
186: int xbee_setupDebugAPI(char *path, int baudrate, char *logfile, char cmdSeq, int cmdTime) {
187:   xbee_hnd xbee = NULL;
188:   int fd, ret;
189:   if ((fd = _open(logfile,_O_WRONLY | _O_CREAT | _O_TRUNC)) == -1) {
190:     fd = 2;
191:   }
192:   ret = xbee_setuplogAPI(path,baudrate,fd,cmdSeq,cmdTime);
193:   if (fd > 2) { /* close fd, as libxbee dup'ed it */
194:     //_close(fd);
195:   }
196:   if (!ret) { /* libxbee started correctly */
197:     xbee = default_xbee;
198:     if (fd == -1) {
199:       xbee_log("Error opening logfile '%s' (errno=%d)... using stderr instead!",logfile,errno);
200:     }
201:   }
202:   return ret;
203: }
204: int xbee_setupDebug(char *path, int baudrate, char *logfile) {
205:   return xbee_setupDebugAPI(path,baudrate,logfile,0,0);
206: }
207:
208: /* These silly little functions are required for VB6
209:    - it freaks out when you call a function that uses va_args... */
210: xbee_con *xbee_newcon_simple(unsigned char frameID, xbee_types type) {
211:   return xbee_newcon(frameID, type);
212: }
213: xbee_con *xbee_newcon_16bit(unsigned char frameID, xbee_types type, int addr) {
214:   return xbee_newcon(frameID, type, addr);
215: }
216: xbee_con *xbee_newcon_64bit(unsigned char frameID, xbee_types type, int addrL, int addrH) {
217:   return xbee_newcon(frameID, type, addrL, addrH);
218: }
219:
220: void xbee_enableACKwait(xbee_con *con) {
221:   con->waitforACK = 1;
222: }
223: void xbee_disableACKwait(xbee_con *con) {
224:   con->waitforACK = 0;
225: }
226:
227: void xbee_enableDestroySelf(xbee_con *con) {
228:   con->destroySelf = 1;
229: }
230:
231: /* for vb6... it will send a message to the given hWnd which can in turn check for a packet */
232: void xbee_callback(xbee_con *con, xbee_pkt *pkt) {
233:   xbee_hnd xbee = default_xbee;
234:
235:   if (!win32_hWnd) {
236:     xbee_log("*** Cannot do callback! No hWnd set... ***");
237:     return;
238:   }
239:   if (!win32_MessageID) {
240:     xbee_log("*** Cannot do callback! No MessageID set... ***");
241:     return;
242:   }
243:
244:   xbee_log("Callback message sent!");
245:   SendMessage(win32_hWnd, win32_MessageID, (int)con, (int)pkt);
246: }
247:
248: /* very simple C function to provide more functionality to VB6 */
249: int xbee_runCallback(int(*func)(xbee_con*,xbee_pkt*), xbee_con *con, xbee_pkt *pkt) {
250:   return func(con,pkt);
251: }
252:
253: void xbee_enableCallbacks(HWND hWnd, UINT uMsg) {
254:   xbee_hnd xbee = default_xbee;
255:   if (!win32_MessageID || win32_MessageID != uMsg) {
```

```
256:     xbee_log("Configuring libxbee to use MessageID = 0x%08X", uMsg);
257:     win32_MessageID = uMsg;
258:   }
259:   if (!win32_hWnd || win32_hWnd != hWnd) {
260:     xbee_log("Configuring libxbee to use hWnd = 0x%08X", hWnd);
261:     win32_hWnd = hWnd;
262:   }
263: }
264:
265: void xbee_attachCallback(xbee_con *con) {
266:   xbee_hnd xbee = default_xbee;
267:
268:   /* setup the callback function */
269:   xbee_log("Setting callback for connection @ 0x%08X",con);
270:   con->callback = xbee_callback;
271: }
272:
273: void xbee_detachCallback(xbee_con *con) {
274:   xbee_hnd xbee = default_xbee;
275:
276:   /* un-setup the callback function */
277:   xbee_log("Unsetting callback for connection @ 0x%08X",con);
278:   con->callback = NULL;
279: }
```