```c
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(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:     perror("xbee_setup():CreateFile()");
49:     xbee_mutex_destroy(xbee.conmutex);
50:     xbee_mutex_destroy(xbee.pktmutex);
51:     xbee_mutex_destroy(xbee.sendmutex);
52:     Xfree(xbee.path);
53:     return -1;
54:   }
55:
56:   GetCommState(xbee.tty, &tc);
57:   tc.BaudRate =          baudrate;
58:   tc.fBinary =           TRUE;
59:   tc.fParity =           FALSE;
60:   tc.fOutxCtsFlow =      FALSE;
61:   tc.fOutxDsrFlow =      FALSE;
62:   tc.fDtrControl =       DTR_CONTROL_DISABLE;
63:   tc.fDsrSensitivity =   FALSE;
64:   tc.fTXContinueOnXoff = FALSE;
65:   tc.fOutX =             FALSE;
66:   tc.fInX =              FALSE;
67:   tc.fErrorChar =        FALSE;
68:   tc.fNull =             FALSE;
69:   tc.fRtsControl =       RTS_CONTROL_DISABLE;
70:   tc.fAbortOnError =     FALSE;
71:   tc.ByteSize =          8;
72:   tc.Parity =            NOPARITY;
73:   tc.StopBits =          ONESTOPBIT;
74:   SetCommState(xbee.tty, &tc);
75:
76:   timeouts.ReadIntervalTimeout = MAXDWORD;
77:   timeouts.ReadTotalTimeoutMultiplier = 0;
78:   timeouts.ReadTotalTimeoutConstant = 0;
79:   timeouts.WriteTotalTimeoutMultiplier = 0;
80:   timeouts.WriteTotalTimeoutConstant = 0;
81:   SetCommTimeouts(xbee.tty, &timeouts);
82:
83:   SetCommMask(xbee.tty, EV_RXCHAR);
84:
85:   return 0;
```

```
 86:  }
 87:
 88:  /* a replacement for the linux select() function... for a serial port */
 89:  static int xbee_select(struct timeval *timeout) {
 90:    int evtMask = 0;
 91:    COMSTAT status;
 92:    int ret;
 93:
 94:    for (;;) {
 95:      /* find out how many bytes are in the Rx buffer... */
 96:      if (ClearCommError(xbee.tty,NULL,&status) && (status.cbInQue > 0)) {
 97:        /* if there is data... return! */
 98:        return 1; /*status.cbInQue;*/
 99:      } else if (timeout && timeout->tv_sec == 0 && timeout->tv_usec == 0) {
100:        /* if the timeout was 0 (return immediately) then return! */
101:        return 0;
102:      }
103:
104:      /* otherwise wait for an Rx event... */
105:      memset(&xbee.ttyovrs,0,sizeof(OVERLAPPED));
106:      xbee.ttyovrs.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
107:      if (!WaitCommEvent(xbee.tty,&evtMask,&xbee.ttyovrs)) {
108:        if (GetLastError() == ERROR_IO_PENDING) {
109:          DWORD timeoutval;
110:          if (!timeout) {
111:            /* behave like the linux function... if the timeout pointer was NULL
112:               then wait indefinately */
113:            timeoutval = INFINITE;
114:          } else {
115:            /* Win32 doesn't give the luxury of microseconds and seconds... just miliseconds! */
116:            timeoutval = (timeout->tv_sec * 1000) + (timeout->tv_usec / 1000);
117:          }
118:          ret = WaitForSingleObject(xbee.ttyovrs.hEvent,timeoutval);
119:          if (ret == WAIT_TIMEOUT) {
120:            /* cause the WaitCommEvent() call to stop */
121:            SetCommMask(xbee.tty, EV_RXCHAR);
122:            /* if a timeout occured, then return 0 */
123:            CloseHandle(xbee.ttyovrs.hEvent);
124:            return 0;
125:          }
126:        } else {
127:          return -1;
128:        }
129:      }
130:      CloseHandle(xbee.ttyovrs.hEvent);
131:    }
132:
133:    /* always return -1 (error) for now... */
134:    return -1;
135:  }
136:
137:  /* this offers the same behavior as non-blocking I/O under linux */
138:  int xbee_write(const void *ptr, size_t size) {
139:    if (!WriteFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrw) &&
140:        (GetLastError() != ERROR_IO_PENDING)) return 0;
141:    if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrw, &xbee.ttyw, TRUE)) return 0;
142:    return xbee.ttyw;
143:  }
144:
145:  /* this offers the same behavior as non-blocking I/O under linux */
146:  int xbee_read(void *ptr, size_t size) {
147:    if (!ReadFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrr) &&
148:        (GetLastError() != ERROR_IO_PENDING)) return 0;
149:    if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrr, &xbee.ttyr, TRUE)) return 0;
150:    return xbee.ttyr;
151:  }
152:
153:  /* this is because Win32 has some weird memory management rules...
154:     - the thread that allocated the memory, must free it... */
155:  void xbee_free(void *ptr) {
156:    if (!ptr) return;
157:    free(ptr);
158:  }
159:
160:  /* enable the debug output to a custom file or fallback to stderr */
161:  int xbee_setupDebugAPI(char *path, int baudrate, char *logfile, char cmdSeq, int cmdTime) {
162:    int fd, ret;
163:    if ((fd = _open(logfile,_O_WRONLY | _O_CREAT | _O_TRUNC)) == -1) {
164:      ret = xbee_setuplogAPI(path,baudrate,2,cmdSeq,cmdTime);
165:    } else {
166:      ret = xbee_setuplogAPI(path,baudrate,fd,cmdSeq,cmdTime);
167:    }
168:    if (fd == -1) {
169:      xbee_log("Error opening logfile '%s' (errno=%d)... using stderr instead...",logfile,errno);
170:    }
```

```
171:     return ret;
172:   }
173:   int xbee_setupDebug(char *path, int baudrate, char *logfile) {
174:     return xbee_setupDebugAPI(path,baudrate,logfile,0,0);
175:   }
176:
177:   /* These silly little functions are required for VB6
178:      - it freaks out when you call a function that uses va_args... */
179:   xbee_con *xbee_newcon_simple(unsigned char frameID, xbee_types type) {
180:     return xbee_newcon(frameID, type);
181:   }
182:   xbee_con *xbee_newcon_16bit(unsigned char frameID, xbee_types type, int addr) {
183:     return xbee_newcon(frameID, type, addr);
184:   }
185:   xbee_con *xbee_newcon_64bit(unsigned char frameID, xbee_types type, int addrL, int addrH) {
186:     return xbee_newcon(frameID, type, addrL, addrH);
187:   }
188:
189:   /* for vb6... it will send a message to the given hWnd which can in turn check for a packet */
190:   void xbee_callback(xbee_con *con, xbee_pkt *pkt) {
191:     win32_callback_info *p = callbackMap;
192:
193:     /* grab the mutex */
194:     xbee_mutex_lock(callbackmutex);
195:
196:     /* see if there is an existing callback for this connection */
197:     while (p) {
198:       if (p->con == con) break;
199:       p = p->next;
200:     }
201:
202:     /* release the mutex (before the SendMessage, as this could take time...) */
203:     xbee_mutex_unlock(callbackmutex);
204:
205:     /* if there is, continue! */
206:     if (p) {
207:       xbee_log("Callback message sent!");
208:       SendMessage(p->hWnd, p->uMsg, (int)con, (int)pkt);
209:     } else {
210:       xbee_log("Callback message NOT sent... Unmapped callback! (con=0x%08X)",con);
211:     }
212:   }
213:
214:   /* very simple C function to provide more functionality to VB6 */
215:   int xbee_runCallback(int(*func)(xbee_con*,xbee_pkt*), xbee_con *con, xbee_pkt *pkt) {
216:     return func(con,pkt);
217:   }
218:
219:   void xbee_attachCallback(xbee_con *con, HWND hWnd, UINT uMsg) {
220:     win32_callback_info *l, *p;
221:
222:     /* grab the mutex */
223:     xbee_mutex_lock(callbackmutex);
224:
225:     l = NULL;
226:     p = callbackMap;
227:
228:     /* see if there is an existing callback for this connection */
229:     while (p) {
230:       if (p->con == con) break;
231:       l = p;
232:       p = p->next;
233:     }
234:     /* if not, then add it */
235:     if (!p) {
236:       p = Xcalloc(sizeof(win32_callback_info));
237:       p->next = NULL;
238:       p->con = con;
239:       if (!l) {
240:         xbee_log("Mapping the first callback...");
241:         callbackMap = p;
242:       } else {
243:         xbee_log("Mapping another callback...");
244:         l->next = p;
245:       }
246:     } else if (xbee.log) {
247:       xbee_log("Updating callback map...");
248:     }
249:     /* setup / update the parameters */
250:     xbee_log("hWnd = [%d]...",hWnd);
251:     xbee_log("uMsg = [%d]...",uMsg);
252:     p->hWnd = hWnd;
253:     p->uMsg = uMsg;
254:
255:     /* setup the callback function */
```

```
256:    con->callback = xbee_callback;
257:
258:    /* release the mutex */
259:    xbee_mutex_unlock(callbackmutex);
260: }
261:
262: void xbee_detachCallback(xbee_con *con) {
263:    win32_callback_info *l = NULL, *p = callbackMap;
264:    xbee_mutex_lock(callbackmutex);
265:
266:    /* see if there is an existing callback for this connection */
267:    while (p) {
268:      if (p->con == con) break;
269:      l = p;
270:      p = p->next;
271:    }
272:    /* if there is, then remove it! */
273:    if (p) {
274:      if (!l) {
275:        callbackMap = NULL;
276:      } else if (l->next) {
277:        l->next = l->next->next;
278:      } else {
279:        l->next = NULL;
280:      }
281:      xbee_log("Unmapping callback...");
282:      xbee_log("hWnd = [%d]...",p->hWnd);
283:      xbee_log("uMsg = [%d]...",p->uMsg);
284:      Xfree(p);
285:    }
286:
287:    con->callback = NULL;
288:
289:    /* release the mutex */
290:    xbee_mutex_unlock(callbackmutex);
291: }
292:
293: /* win32 equivalent of unix gettimeofday() */
294: int gettimeofday(struct timeval *tv, struct timezone *tz) {
295:    if (tv) {
296:      struct _timeb timeb;
297:
298:      _ftime(&timeb);
299:      tv->tv_sec = timeb.time;
300:      tv->tv_usec = timeb.millitm * 1000;
301:    }
302:    /* ignore tz for now */
303:    return 0;
304: }
```