```c
/*
    libxbee - a C library to aid the use of Digi’s Series 1 XBee modules
              running in API mode (AP=2).

    Copyright (C) 2009  Attie Grande (attie@attie.co.uk)

    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 3 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, see <http://www.gnu.org/licenses/>.
*/

/* ################################################################# */
/* ### Win32 Code ################################################## */
/* ################################################################# */

/*  this file contains code that is used by Win32 ONLY */
#ifndef _WIN32
#error "This file should only be used on a Win32 system"
#endif

#include "win32.h"
#include "win32.dll.c"

/* this is because Win32 has some weird memory management rules...
   - the thread that allocated the memory, must free it... */
void xbee_free(void *ptr) {
  if (!ptr) return;
  free(ptr);
}

/* These silly little functions are required for VB6
   - it freaks out when you call a function that uses va_args... */
xbee_con *xbee_newcon_simple(unsigned char frameID, xbee_types type) {
  return xbee_newcon(frameID,type);
}
xbee_con *xbee_newcon_16bit(unsigned char frameID, xbee_types type, int addr) {
  return xbee_newcon(frameID,type, addr);
}
xbee_con *xbee_newcon_64bit(unsigned char frameID, xbee_types type, int addrL, int addrH) {
  return xbee_newcon(frameID,type,addrL,addrH);
}

int init_serial(int baudrate) {
  int chosenbaud;
  DCB tc;
  int evtMask;
  COMMTIMEOUTS timeouts;

  /* open the serial port */
  xbee.tty = CreateFile(TEXT(xbee.path),
                        GENERIC_READ | GENERIC_WRITE,
                        0,    /* exclusive access */
                        NULL, /* default security attributes */
                        OPEN_EXISTING,
                        FILE_FLAG_OVERLAPPED,
                        NULL);
  if (xbee.tty == INVALID_HANDLE_VALUE) {
    perror("xbee_setup():CreateFile()");
    xbee_mutex_destroy(xbee.conmutex);
    xbee_mutex_destroy(xbee.pktmutex);
    xbee_mutex_destroy(xbee.sendmutex);
    Xfree(xbee.path);
    return -1;
  }

  GetCommState(xbee.tty, &tc);
  tc.BaudRate =          baudrate;
  tc.fBinary =           TRUE;
  tc.fParity =           FALSE;
  tc.fOutxCtsFlow =      FALSE;
  tc.fOutxDsrFlow =      FALSE;
  tc.fDtrControl =       DTR_CONTROL_DISABLE;
  tc.fDsrSensitivity =   FALSE;
  tc.fTXContinueOnXoff = FALSE;
  tc.fOutX =             FALSE;
  tc.fInX =              FALSE;
  tc.fErrorChar =        FALSE;
  tc.fNull =             FALSE;
  tc.fRtsControl =       RTS_CONTROL_DISABLE;
  tc.fAbortOnError =     FALSE;
  tc.ByteSize =          8;
  tc.Parity =            NOPARITY;
  tc.StopBits =          ONESTOPBIT;
  SetCommState(xbee.tty, &tc);

  timeouts.ReadIntervalTimeout = MAXDWORD;
  timeouts.ReadTotalTimeoutMultiplier = 0;
  timeouts.ReadTotalTimeoutConstant = 0;
  timeouts.WriteTotalTimeoutMultiplier = 0;
  timeouts.WriteTotalTimeoutConstant = 0;
  SetCommTimeouts(xbee.tty, &timeouts);

  SetCommMask(xbee.tty, EV_RXCHAR);

  return 0;
}

/* a replacement for the linux select() function... for a serial port */
static int xbee_select(struct timeval *timeout) {
  int evtMask = 0;
  COMSTAT status;
  int ret;

  for (;;) {
    /* find out how many bytes are in the Rx buffer... */
    if (ClearCommError(xbee.tty,NULL,&status) && (status.cbInQue > 0)) {
      /* if there is data... return! */
      return 1; /*status.cbInQue;*/
    } else if (timeout && timeout->tv_sec == 0 && timeout->tv_usec == 0) {
      /* if the timeout was 0 (return immediately) then return! */
      return 0;
    }

    /* otherwise wait for an Rx event... */
    memset(&xbee.ttyovrs,0,sizeof(OVERLAPPED));
    xbee.ttyovrs.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
    if (!WaitCommEvent(xbee.tty,&evtMask,&xbee.ttyovrs)) {
      if (GetLastError() == ERROR_IO_PENDING) {
        DWORD timeoutval;
        if (!timeout) {
          /* behave like the linux function... if the timeout pointer was NULL
             then wait indefinately */
          timeoutval = INFINITE;
        } else {
          /* Win32 doesn’t give the luxury of microseconds and seconds... just miliseconds! */
          timeoutval = (timeout->tv_sec * 1000) + (timeout->tv_usec / 1000);
        }
        ret = WaitForSingleObject(xbee.ttyovrs.hEvent,timeoutval);
        if (ret == WAIT_TIMEOUT) {
          /* cause the WaitCommEvent() call to stop */
          SetCommMask(xbee.tty, EV_RXCHAR);
          /* if a timeout occured, then return 0 */
          CloseHandle(xbee.ttyovrs.hEvent);
          return 0;
        }
      } else {
        return -1;
      }
    }
    CloseHandle(xbee.ttyovrs.hEvent);
  }

  /* always return -1 (error) for now... */
  return -1;
}

/* this offers the same behavior as non-blocking I/O under linux */
int xbee_write(const void *ptr, size_t size) {
  if (!WriteFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrw) &&
      (GetLastError() != ERROR_IO_PENDING)) return 0;
  if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrw, &xbee.ttyw, TRUE)) return 0;
  return xbee.ttyw;
}

/* this offers the same behavior as non-blocking I/O under linux */
int xbee_read(void *ptr, size_t size) {
  if (!ReadFile(xbee.tty, ptr, size, NULL, &xbee.ttyovrr) &&
      (GetLastError() != ERROR_IO_PENDING)) return 0;
  if (!GetOverlappedResult(xbee.tty, &xbee.ttyovrr, &xbee.ttyr, TRUE)) return 0;
  return xbee.ttyr;
}
```

```c

const char *xbee_svn_version(void) {
  /* need to work out a way to get the SVN version into this function... */
  return "Win32";
}
```