

C listing: PLINKC.C

```

/*****
/*
/*          P L I N K C . C
/*
/*-----*/
/*   Task           : Transfers files over the parallel interface.
/*-----*/
/*   Author          : Michael Tischer
/*   Developed on    : 09/27/91
/*   Last update     : 04/07/95
/*-----*/
/*   Memory model    : SMALL
*****/

```

```
/*== Add include files =====*/
```

```

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <setjmp.h>
#include <string.h>
#ifdef __TURBOC__
    #include <dir.h>
    #include <ctype.h>
#endif
/* Turbo C compiler? */
/* Include directory functions */
/* for toupper() */

```

```
/*== Type definitions =====*/
```

```

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef struct {
    BYTE
    Token;
/* Define a byte */
/* Create a WORD */
/* Block transfer header */

```

```

        unsigned int Len;
    } BHEADER;

/*== Assembler module functions =====*/

extern void IntrInstall( int far * escape_flag,
                        WORD far * timeout_count );
extern void IntrRemove( void );
extern void EscapeDirect( int disconnect );

/*== Constants =====*/

#define ONESEC      18                      /* One second */
#define TENSEC     182                     /* Ten seconds */
#define TO_DEFAULT  TENSEC                 /* Time out default value */

#define TRUE        ( 0 == 0 )
#define FALSE       ( 0 == 1 )
#define MAXBLOCK    4096                   /* 4K maximum block size */

/*-- Constants for transfer protocol -----*/

#define ACK          0x00                   /* Acknowledge */
#define NAK          0xFF                   /* Non-Acknowledge */
#define MAX_TRY      5                     /* Maximum number of tries */

/*-- Tokens for communication between sender and receiver -----*/

#define TOK_DATSTART 0                      /* Start of file data */
#define TOK_DATNEXT  1                      /* Next file data block */
#define TOK_DATEND    2                      /* End file data transfer */
#define TOK_ENDIT     3                      /* End program */

```

```

#define TOK_ESCAPE      4          /* <Esc> pressed on remote computer */

/*-- Codes for LongJump call -----*/

#define LJ_OKSENDER     1          /* All data sent successfully */
#define LJ_OKRECD       2          /* All data received successfully */
#define LJ_TIMEOUT      3          /* Time out: No response */
#define LJ_ESCAPE       4          /* <Esc> pressed on local computer */
#define LJ_REMESCPE     5          /* <Esc> pressed on remote computer */
#define LJ_DATA         6          /* Communication error */
#define LJ_NOLINK       7          /* No link */
#define LJ_NOPAR        8          /* No interface */
#define LJ_PARA         9          /* Invalid call parameters */

/*== Macros =====*/

/*-- The lower three bits of the input register are not allocated ---*/
/*-- and can be set to 1 or 0 by the computer, so they are masked ---*/
/*-- by GetB(). -----*/

#ifdef __TURBOC__
/* Turbo C compiler? */
#define GetB() ( inportb( InpPort ) & 0xF8 )
#define PutB( Was ) outportb( OutPort, Was )
#define DIRSTRUCT struct ffbldk
#define FINDFIRST( path, buf, attr ) findfirst( path, buf, attr )
#define FINDNEXT( buf ) findnext( buf )
#define DFILENAME ff_name
#else
/* No --> Microsoft C */
#define GetB() ( inp( InpPort ) & 0xF8 )
#define PutB( Was ) outp( OutPort, Was )
#define DIRSTRUCT struct find_t
#define FINDFIRST( path, buf, attr ) _dos_findfirst(path, attr, buf)

```

```

#define FINDNEXT( buf )                _dos_findnext( buf )
#define DFILENAME                      name
#endif

#ifdef MK_FP                            /* Macro MK_FP already defined? */
    #undef MK_FP                       /* Yes --> Undefine macro */
#endif

#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))

/*== Global variables =====*/
int      InpPort;                      /* Input port address */
int      OutPort;                     /* Output port address */
int      Escape = 0;                  /* <Esc> not pressed */
WORD     Timeout = TO_DEFAULT;        /* Selected time out value */
WORD     TO_Count;                    /* Counter for time out */
jmp_buf  ReturnToEnd;                 /* Return address for end */
BYTE     *BlockBuf;                   /* Buffer for passing a block */
FILE     *FiVar = NULL;               /* File variable for file being processed */

/*****
/* GetPortAdr: Initializes a parallel interface's port addresses in
/*      the global variables INPPORT and OUTPORT.
/* Input      : LPTNUM = Parallel interface number (1-4)
/* Output     : TRUE if interface is valid
/* Global vars. : InpPort/W, OutPort/W
/* Info       : The base addresses of up to 4 parallel interfaces
/*      lie in the four memory words starting at
/*      0040:0008H.
*****/

```

```

int GetPortAdr( int LptNum )
{
    /* Read port addresses from BIOS variable segment */
    OutPort = *( WORD far * ) MK_FP( 0x0040, 6 + LptNum * 2 );
    if ( OutPort != 0 )
        /* Interface available? */
        {
            /* Yes */
            InpPort = OutPort + 1;
            /* Input register address */
            return TRUE;
            /* End error-free */
        }
    else
        return FALSE;
    /* Error: Interface not available */
}

```

```

/*****
/* Port_Init      : Initializes the registers needed for transfer.      */
/* Input          : SENDER = TRUE if sender, FALSE if receiver          */
/* Output         : TRUE if register initializes successfully            */
/* Global vars.   : InpPort/R, OutPort/R                                */
/* Info           : The asymmetry (send 00010000, wait for 00000000)    */
/*                : occurs because of signal inversion. Normally the    */
/*                : input and output registers contain the desired      */
/*                : values, but initialization is needed after an       */
/*                : aborted transfer, or when restarting.               */
*****/

```

```

int Port_Init( int Sender )
{
    EscapeDirect( TRUE );
    if ( Sender )
        /* Release through Escape time out */
        /* Device = Sender? */
        {
            TO_Count = Timeout * 5;
            /* Start time out counter */
            PutB( 0x10 );
            /* Send: 00010000b */
            while ( ( GetB() != 0x00 ) && TO_Count )
                /* Wait for 00000000b */
        }
}

```

```

    ;
}
else
{
    TO_Count = Timeout * 5; /* Start time out counter */
    while ( ( GetB() != 0x00 ) && TO_Count ) /* Wait for 00000000b */
    ;
    PutB( 0x10 ); /* Send: 00010000b */
}
EscapeDirect( FALSE ); /* No time out released on Escape */
return ( TO_Count != 0 ); /* End initialization */
}

```

```

/*****
/* SendABYTE      : Sends a byte to the remote computer in two parts, */
/*                  then checks the result.                             */
/* Input          : B2Send = Byte to be sent                          */
/* Output         : Transfer successful? (0 = error, -1 = O.K.)         */
/* Global vars.   : Timeout/R, InpPort/R, OutPort/R (in macros)        */
*****/

```

```

int SendABYTE( BYTE B2Send )
{
    BYTE RcvdB; /* Received byte */

    /*-- Send lower nibble -----*/

    TO_Count = Timeout; /* Initialize time out counter */
    PutB( B2Send & 0x0F ); /* Sending, clear BUSY */
    while ( ( ( GetB() & 128 ) == 0 ) && TO_Count ) /* Wait for message */
    ;
    if ( TO_Count == 0 ) /* Time out error? */

```

```

    longjmp( ReturnToEnd, LJ_TIMEOUT );                /* Cancel transfer */

RcvdB = ( GetB() >> 3 ) & 0x0F;                        /* Bits 3-6 in 0-3 */

/*-- Send upper nibble -----*/

TO_Count = Timeout;                                   /* Initialize time out counter */
PutB( ( B2Send >> 4 ) | 0x10 );                       /* Sending, set BUSY */
while ( ( ( GetB() & 128 ) != 0 ) && TO_Count ) /* Wait for message */
    ;

if ( TO_Count == 0 )                                  /* Time out error? */
    longjmp( ReturnToEnd, LJ_TIMEOUT );                /* Cancel transfer */

RcvdB = RcvdB | ( ( GetB() << 1 ) & 0xF0 );           /* Bits 3-6 in 4-7 */
return ( B2Send == RcvdB );                           /* Byte sent correctly? */
}

/*****
/* ReceiveAByte : Receives a two part byte from remote computer, and */
/*               sends returned parts for testing.                      */
/* Input        : None                                                  */
/* Output       : Received byte                                         */
/* Global vars. : Timeout/R, InpPort/R, OutPort/R (in macros)         */
*****/

BYTE ReceiveAByte( void )
{
    BYTE LoNib, HiNib;                                                /* Received nibbles */

    /*-- Receive and re-send lowest nibble -----*/

```

```

TO_Count = Timeout;                                /* Initialize time out counter */
while ( ( ( GetB() & 128 ) == 0 ) && TO_Count )/* Wait until BUSY 1 */
;

if ( TO_Count == 0 )                                /* Time out error? */
    longjmp( ReturnToEnd, LJ_TIMEOUT );            /* Cancel transfer */

LoNib = ( GetB() >> 3 ) & 0x0F;                    /* Bits 3-6 in 0-3 */
PutB( LoNib );                                      /* Re-send */

/*-- Receive and re-send highest nibble -----*/

TO_Count = Timeout;                                /* Initialize time out counter */
while ( ( ( GetB() & 128 ) != 0 ) && TO_Count )/* Wait until BUSY 0 */
;

if ( TO_Count == 0 )                                /* Time out error? */
    longjmp( ReturnToEnd, LJ_TIMEOUT );            /* Cancel transfer */

HiNib = ( GetB() << 1 ) & 0xF0;                    /* Bits 3-6 in 4-7 */
PutB( ( HiNib >> 4 ) | 0x10 );                    /* Re-sending, set BUSY */

return( LoNib | HiNib );                            /* Received byte */
}

/*****
/* SendABlock: Sends a data block.                                */
/* Input      : TOKEN = Token for receiver                        */
/*            : TRANUM = Number of bytes to be transferred      */
/*            : DPTR  = Pointer to buffer containing data       */
/* Output     : None, jumps immediately to an error handler if  */
/*            : an error occurs.                                  */

```



```
/******
```

```
void SendABlock( BYTE Token, int TraNum, void *DPtr )
{
    BHEADER header;          /* Header for placing tokens and numbers */
    BYTE    *bptr,           /* Pointer to current byte to be sent */
            RcvrEscape;      /* <Esc> pressed on remote? */
    int      ok,              /* Error flag */
            i,                /* Loop counter */
            try;              /* Remaining number of tries */

    if ( Escape )             /* <Esc> pressed on local computer? */
    {
        Token = TOK_ESCAPE;   /* Yes --> Send Escape token */
        TraNum = 0;
    }

    /*-- Send header first -----*/

    header.Token = Token;     /* Create header */
    header.Len = TraNum;

    for ( try = MAX_TRY; try; --try ) /* Make MAX_TRY access attempts */
    {
        ok = TRUE;           /* Send on error-free transfer */
        for ( bptr = (BYTE *) &header, i = sizeof( header); i; --i )
            ok = ok & SendAByte( *bptr++ ); /* Send a byte */

        ok = ok & SendAByte( (BYTE) (ok ? ACK : NAK) ); /* Confirmation */
        if ( ok )             /* Everything transfer successfully? */
            break;             /* Yes --> No need to try again */
    }
}
```

```

if ( try == 0 ) /* Could the header be sent successfully? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */

if ( Token == TOK_ESCAPE ) /* Was an Escape message sent? */
longjmp( ReturnToEnd, LJ_ESCAPE ); /* Yes --> Cancel transfer */

/*-- Send the data block itself -----*/

if ( TraNum ) /* Length > 0? */
{
    for ( try = MAX_TRY; try; -- try ) /* Make MAX_TRY attempts */
    {
        ok = TRUE; /* Send on error-free transfer */
        for ( bptr = (BYTE *) DPtr, i = TraNum; i; --i )
            ok = ok & SendABYTE( *bptr++ ); /* Send byte and read status */

        ok = ok & SendABYTE( (BYTE) (ok ? ACK : NAK) ); /* Confirmation */
        if ( ok ) /* Everything transfer successfully? */
            break; /* Yes --> No need to try again */
    }

    if ( try == 0 ) /* Could data block be sent successfully? */
        longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
}

/*-- Read Escape byte from receiver -----*/

for ( try = MAX_TRY; try; -- try ) /* Enable tries */
{
    RcvrEscape = ReceiveABYTE(); /* Read remove Escape status */
    if ( RcvrEscape == (BYTE) TRUE || RcvrEscape == (BYTE) FALSE )
        break; /* Receive Escape status */
}

```

```

    }
    if ( try == 0 )                /* Was the Escape status received? */
        longjmp( ReturnToEnd, LJ_DATA );        /* No --> Cancel transfer */

    if ( RcvrEscape )              /* <Esc> from remote computer? */
        longjmp( ReturnToEnd, LJ_REMESCPE );    /* Yes --> Cancel transfer */
}

/*****
/* ReceiveABlock: Receives a data block.
/* Input      : TOKEN = Pointer to variable containing tokens
/*             LEN    = Pointer to variable containing length
/*             DPTR   = Pointer to buffer containing data
/* Output     : None, jumps immediately to an error handler if
/*             an error occurs.
/* Info      : Buffer must be allocated using MAXBLOCK, since block
/*             length cannot usually be anticipated.
*****/

void ReceiveABlock( BYTE *Token, int *Len, void *DPtr )
{
    BHEADER header;                /* Header for storing tokens and numbers */
    BYTE *bptr;                    /* Floating pointer for receive buffer */
    int ok,                        /* Error flag */
        i,                        /* Loop counter */
        try,                      /* Remaining number of tries */
        EscapeStatus;             /* For storing current Escape status */

    /*-- Receive header first -----*/

    for ( try = MAX_TRY; try; -- try ) /* Make MAX_TRY access attempts */
    {

```

```

for ( bptr = (BYTE *) &header, i = sizeof(header); i; --i )
    *bptr++ = ReceiveAByte( );

if ( ReceiveAByte() == ACK ); /* All bytes received successfully? */
    break;                    /* Yes --> No need to try again */
}

if ( try == 0 )                /* Header received successfully? */
    longjmp( ReturnToEnd, LJ_DATA );    /* No --> Cancel transfer */

if ( ( *Token = header.Token ) == TOK_ESCAPE )    /* Sender Escape? */
    longjmp( ReturnToEnd, LJ_REMESCPE );    /* Yes --> Cancel transfer */

/*-- Header was O.K., now receive the data block itself -----*/

if ( ( *Len = header.Len ) != 0 )                /* No data block? */
{
    /* No */
    for ( try = MAX_TRY; try; -- try )/* Make MAX_TRY access attempts */
    {
        for ( bptr = (BYTE *) DPtr, i = header.Len; i; --i )
            *bptr++ = ReceiveAByte( );

        if ( ReceiveAByte() == ACK );    /* Bytes received successfully? */
            break;                    /* Yes --> No need to try again */
    }

    if ( try == 0 )                /* Block received successfully? */
        longjmp( ReturnToEnd, LJ_DATA );    /* No --> Cancel transfer */
}

/*-- Send current Escape status to the remote computer -----*/

```

```

EscapeStatus = Escape;                                /* Note status */
for ( try = MAX_TRY; try; -- try )                    /* Enable access attempts */
{
    if ( SendAByte( (BYTE) (EscapeStatus != 0) ) )     /* Sent? */
        break;                                       /* Yes --> No need to try again */
}

if ( try == 0 )                                        /* Could Escape status be sent? */
    longjmp( ReturnToEnd, LJ_DATA );                 /* No --> Cancel transfer */

if ( EscapeStatus )                                   /* <Esc> pressed on this computer? */
    longjmp( ReturnToEnd, LJ_ESCAPE );                /* Yes --> Cancel transfer */
}

/*****
/* SendAFile : Sends a file.                                */
/* Input      : NAME = Pointer to buffer containing filenames */
/* Output     : None                                         */
*****/

void SendAFile( char *Name )
{
    int          Status;                                /* Send status */
    WORD         WdsRead;                               /* Number of bytes read */
    unsigned long BytSent;                              /* Number of bytes sent */

    printf( "%-13s", Name );
    FiVar = fopen( Name, "rb" );                        /* Open file */
    SendABlock( TOK_DATSTART, strlen(Name)+1, Name );  /* Send filename */

    /*-- Transfer file contents -----*/
    BytSent = 0;

```



```

char                Name[13];                                /* Filename */

ReceiveABlock( &Token, &Len, BlockBuf );
if ( Token == TOK_DATSTART )
{
    strcpy( Name, BlockBuf );
    FiVar = fopen( Name, "wb" );                            /* Open (create) file */
    printf( "%-13s", Name );

    /*-- Receive file contents -----*/

    BytSent = 0;
    do
    {
        ReceiveABlock( &Token, &Len, BlockBuf );           /* Receive block */
        if ( Token == TOK_DATNEXT )                         /* Next data block? */
        {
            /* Yes */
            fwrite( BlockBuf, 1, Len, FiVar );                /* Write block */
            BytSent += Len;
            printf( "\r%-13s (%ld)", Name, BytSent );
        }
    }
    while ( Token == TOK_DATNEXT );
    fclose( FiVar );                                         /* Close file */
    FiVar = NULL;                                           /* File is closed */
    printf( "\n" );
}
return Token;                                              /* Return error status */
}

/*****
/*                M A I N      P R O G R A M                */
*/

```

```
/* **** */
```

```
void main( int argc, char *argv[] )
{
    DIRSTRUCT SRec;                /* Structure for directory search */
    BYTE      Sender;              /* Transfer mode (Send, Receive) */
    int       sjStatus,            /* Longjump code */
            LptNum,                /* Interface number */
            i,                      /* Loop counter */
            DirFound;              /* For directory search */
}
```

```
static char *ScnMesg[ 9 ] =
{
    "DONE: All files sent successfully.",
    "DONE: All files received successfully.",
    "ERROR: Time out, remote system not responding.",
    "DONE: <Esc> key pressed.",
    "DONE: <Esc> key pressed on remote computer.",
    "ERROR: Hardware problem (check cable and interface).",
    "ERROR: No contact with remote computer.",
    "ERROR: Interface you requested not found.",
    "ERROR: Invalid or unknown parameters." };
```

```
printf( "\n\nParallel Interface File Transfer Program    " );
printf( "(c) 1992 by Michael Tischer\n" );
printf( "===== " );
printf( "=====\n\n" );
```

```
if ( strcmp( argv[ 1 ], "/" ) == 0 )        /* Display syntax only? */
{
    printf( "Syntax: plinkc [/Pn] [/Tnn] [filename]\n" );
    printf( "      |           |\n" );
    printf( "      Interface   Time out " );
}
```



```

    exit ( 0 );
}

sjStatus = setjmp( ReturnToEnd );          /* Return address to end */
if ( sjStatus )                            /* Longjmp called? */
{                                           /* Yes */
    IntrRemove( );                        /* Disable interrupt handler */
    if ( FiVar )                          /* Any files still open? */
        fclose( FiVar );
    free( BlockBuf );                     /* Release allocated buffer memory */
    printf( "\n\n%s\n", ScnMesg[ sjStatus - 1 ] );
    exit( 0 );
}

BlockBuf = malloc( MAXBLOCK );             /* Generate data buffer */
IntrInstall( &Escape, &TO_Count );        /* Interrupt handler init */

/*-- Set default values and interpret command line -----*/

Sender = FALSE;                           /* Default is Receiver */
LptNum = 1;                               /* Default is LPT1 */

for ( i = 1; i < argc; i++ )
{
    if ( argv[i][0] == '/' )                /* Parameter */
    {
        switch ( toupper( argv[i][1] ) )
        {
            case 'T' : Timeout = (atol( &argv[i][2] ) * TENSEC) / 10;
                        if ( Timeout == 0 )
                            longjmp( ReturnToEnd, LJ_PARA );        /* Invalid */
                        break;

```

```

        case 'P' : LptNum = argv[i][2] - 48;                /* Interface */
                    if ( LptNum == 0 || LptNum > 4 )
                        longjmp( ReturnToEnd, LJ_PARA );    /* Invalid */
                    break;
        default : longjmp( ReturnToEnd, LJ_PARA ); /* Unknown params */
                    break;
    }
    argv[i][0] = '\0';                                     /* Clear argument */
}
else                                                       /* No parameters, must be a filename */
    Sender = TRUE;                                         /* Sender */
}

/*-- Start transfer -----*/

if ( GetPortAdr(LptNum) == FALSE )           /* Does the interface exist? */
    longjmp( ReturnToEnd, LJ_NOPAR );        /* No --> Error */

if ( Port_Init( Sender ) == FALSE )          /* Create link */
    longjmp( ReturnToEnd, LJ_NOLINK );       /* Impossible, error */

if ( Sender )                                 /* Sender? */
{
    printf( "Sending over LPT%d:\n\n", LptNum );

    /*-- Transfer all data -----*/

    for ( i = 1; i < argc; i++ )              /* Execute command line */
    {
        if ( argv[i][0] != '\0' )             /* Filename? */
        {                                     /* Yes */
            DirFound = FINDFIRST( argv[i], &SRec, 0);

```

```

while ( !DirFound )
{
    if ( SRec.DFILENAME[0] != '.' )
        SendAFile( SRec.DFILENAME );           /* Transfer file */
    DirFound = FINDNEXT( &SRec );
}
}
SendABlock( TOK_ENDIT, 0 , NULL );             /* All files sent? */
longjmp( ReturnToEnd, LJ_OKSENDER );
}
else                                           /* No --> Receiver */
{
    printf( "Receiving over LPT%d:\n\n", LptNum );
    while ( ReceiveAFile() != TOK_ENDIT )      /* Receive data until */
        ;                                       /* ENDIT token      */
    longjmp( ReturnToEnd, LJ_OKRECD );
}
}

```