TFTP Data Transfer Test between Rpi3B running Ultibo & Rpi2B 01/12/17

Retrieve the code from github. ***"git clone https://github.com/pjde/ultibo-tftp.git"***

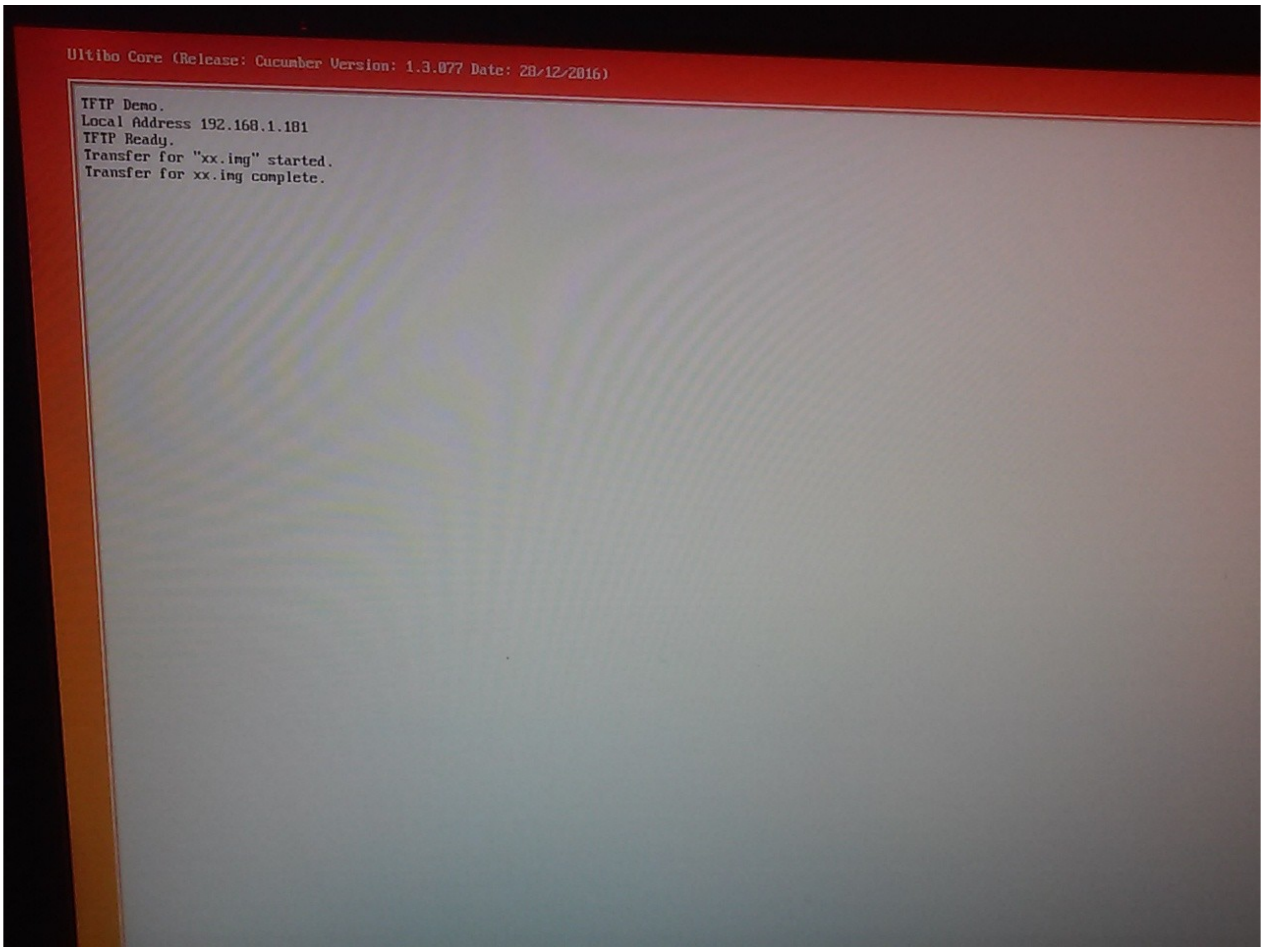
Add the fpc ultibo edition path to your PATH..

"export PATH=/home/pi/ultibo/core/fpc/bin:$PATH"

Compile the Pascal code uTFTP.pas found at Appendix A and TFTPTest.lpr found at Appendix B.

"fpc -B -Tultibo -Parm -CpARMV7A -WpRPI3B @/home/pi/ultibo/core/fpc/bin/rpi3.cfg -O2 TFTPTest.lpr"

Boot the Rpi3 with TFTP program.



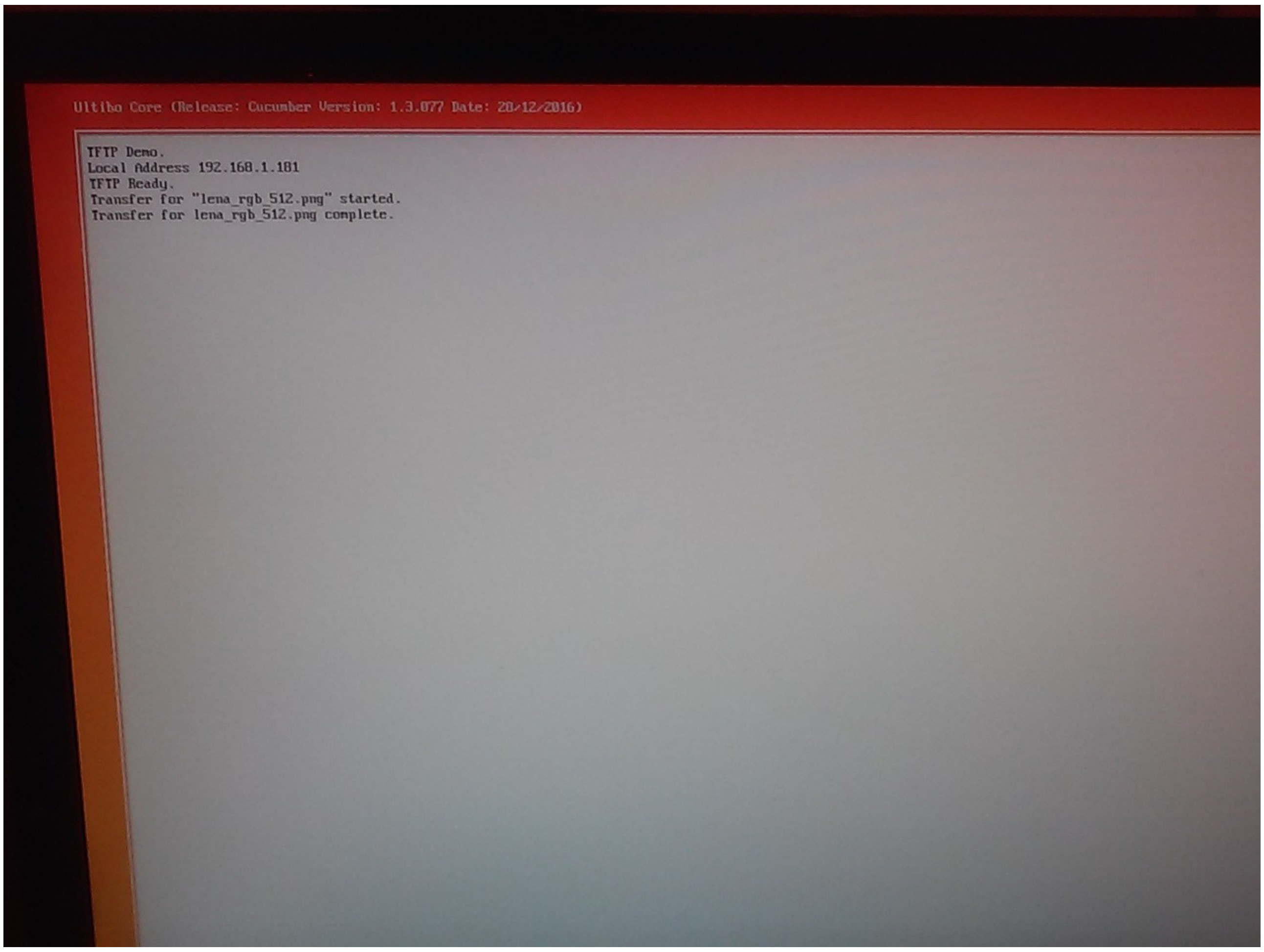
```
Ultibo Core (Release: Cucumber Version: 1.3.077 Date: 28/12/2016)

TFTP Demo.
Local Address 192.168.1.181
TFTP Ready.
Transfer for "xx.img" started.
Transfer for xx.img complete.
```

```
pi@raspberrypi2-142:~/test_ultibo/ultibo-tftp $ tftp 192.168.1.181
tftp> binary
tftp> trace
Packet tracing on.
```

```
tftp> put kernel7.img xx.img
sent WRQ <file=xx.img, mode=octet>
received ACK <block=0>
sent DATA <block=1, 512 bytes>
received ACK <block=1>
sent DATA <block=2, 512 bytes>

sent DATA <block=4518, 32 bytes>
received ACK <block=4518>
Sent 2312736 bytes in 160.0 seconds
```



```
Ultibo Core (Release: Cucumber Version: 1.3.077 Date: 20/12/2016)

TFTP Demo.
Local Address 192.168.1.181
TFTP Ready.
Transfer for "lena_rgb_512.png" started.
Transfer for lena_rgb_512.png complete.
```

```
pi@raspberrypi2-142:~/test_ultibo/LiBc $ tftp 192.168.1.181
tftp> binary
tftp> put lena_rgb_512.png
Sent 476235 bytes in 54.4 seconds
```

Appendix A

```
unit uTFTP;

{$mode objfpc}{$H+}

{ Simple Trival FTP Server based on RFC 1350

  Ultibo (C) 2015 - SoftOz Pty Ltd.                    LGPLv2.1 with static linking exception
  FPC (c) 1993-2015 Free Pascal Team.                    Modified Library GNU Public License
  Lazarus (c) 1993-2015 Lazarus and Free Pascal Team.      Modified Library GNU Public License

  Other bits (c) 2016 pjde                               LGPLv2.1 with static linking exception
}

interface

uses
  GlobalConfig,
  GlobalConst,
  Platform,
  Threads,
  SysUtils,
  Classes,
  Winsock2;

const
  // TFTP opcodes
  TFTP_RRQ                   = 1;
  TFTP_WRQ                   = 2;
  TFTP_DATA                  = 3;
  TFTP_ACK                   = 4;
  TFTP_ERROR                 = 5;
  TFTP_OACK                  = 6;

  // TFTP error codes
  erUNDEFINED                = 0;
  erFILE_NOT_FOUND           = 1;
  erACCESS_VIOLATION         = 2;
  erALLOCATION_EXCEEDED      = 3;
  erILLEGAL_OPERATION        = 4;
  erUNKNOWN_TRANSFER_ID      = 5;
  erFILE_ALREADY_EXISTS      = 6;
  erNO_SUCH_USER             = 7;
  erOPTION_NEGOTIATION_FAILED = 8;

type
  TMsgEvent = procedure (Sender : TObject; s : string);

  TTFTPListener = class;

  { TFTPTransferThread }

  TFTPTransferThread = class (TWinsock2UDPServerThread)
    FStream : TMemoryStream;
    function ReadByte : byte;
    function ReadWord : Word;
    function ReadString : string;
    function Remaining : int64;
    constructor Create (aListener : TWinsock2UDPServer);
    destructor Destroy; override;
  end;


  { TTransfer }

  TTransfer = class
    FileName : string;
    FStream : TMemoryStream;
    FListener : TTFTPListener;
    Op : word;
    TID : Word;
    BlockNo : Word;
    SockAddr : PSockAddr;
    SockAddrLength : integer;
    constructor Create (aListener : TTFTPListener);
    destructor Destroy; override;
  end;

  { TTFTPListener }

  TTFTPListener = class (TWinsock2UDPListener)
    FOnMsg : TMsgEvent;
    FRebootOnImg : boolean;
  private
    TxStream : TMemoryStream;
    procedure SetOnMsg (Value : TMsgEvent);
    procedure AddByte (b : byte);
    procedure AddWord (w : Word);
    procedure AddString (s : string);
  protected
    Transfers : TList;
    function GetTransfer (byID : Word) : TTransfer;
    procedure RemoveTransfer (byID : Word); overload;
    procedure RemoveTransfer (byTransfer : TTransfer); overload;
     procedure DoMsg (s : string);
    function DoExecute (aThread : TWinsock2UDPServerThread) : Boolean; override;
    procedure DoCreateThread (aServer : TWinsock2UDPServer; var aThread :
TWinsock2UDPServerThread);
  public
    constructor Create;
    destructor Destroy; override;
    procedure SendError (aTransfer : TTransfer; ErrCode : Word; ErrMsg : string); overload;
    procedure SendError (aServer : TWinsock2UDPServer; ErrCode : Word; ErrMsg : string);
overload;
    procedure SendAck (aTransfer : TTransfer; BlockNo : Word);
    procedure SendDataBlock (aTransfer : TTransfer; BlockNo : Word);
    property OnMsg : TMsgEvent read FOnMsg write SetOnMsg;
    property RebootOnImg : boolean read FRebootOnImg write FRebootOnImg;
  end;


procedure SetOnMsg (MsgProc : TMsgEvent);
```

```
var
  TFTP : TTFTPListener = nil;

implementation

procedure SetOnMsg (MsgProc : TMsgEvent);
begin
  if Assigned (TFTP) then TFTP.OnMsg := MsgProc;
end;

{ TTransfer }

constructor TTransfer.Create (aListener : TTFTPListener);
begin
  FStream := TMemoryStream.Create;
  FListener := aListener;
  FileName := '';
  Op := 0;
  TID := 0;
  BlockNo := 0;
  SockAddr := nil;
  SockAddrLength := 0;
end;

destructor TTransfer.Destroy;
begin
  FStream.Free;
  if SockAddr <> nil then
    begin
      if FListener <> nil then FListener.ReleaseAddress (SockAddr, SockAddrLength);
      SockAddr := nil;
    end;
  inherited Destroy;
end;

{ TFTPTransferThread }

function TFTPTransferThread.ReadByte: byte;
begin
  Result := 0;
  if Remaining > 0 then FStream.Read (Result, 1);
end;

function TFTPTransferThread.ReadWord : Word;
begin
  Result := ReadByte * $100 + ReadByte;
end;

function TFTPTransferThread.ReadString : string;
var
  ch : Char;
```

```
begin
  Result := '';
  ch := '~';
  while (Remaining > 0) and (ch <> #0) do
    begin
      FStream.Read (ch, 1);
      if ch <> #0 then Result := Result + ch;
    end;
end;

function TFTPTransferThread.Remaining: int64;
begin
  Result := FStream.Size - FStream.Position;
end;

constructor TFTPTransferThread.Create (aListener : TWinsock2UDPServer);
begin
  inherited Create (aListener);
  FStream := TMemoryStream.Create;
end;

destructor TFTPTransferThread.Destroy;
begin
  FStream.Free;
  inherited Destroy;
end;

constructor TTFTPListener.Create;
begin
  inherited Create;
  Threads.Min := 5;
  Threads.Max := 10;
  BufferSize := 1024;
  BoundPort := 69;
  Transfers := TList.Create;
  FRebootOnImg := true;
  FOnMsg := nil;
  { Define custom thread }
  OnCreateThread := @DoCreateThread;
  TxStream := TMemoryStream.Create;
end;


destructor TTFTPListener.Destroy;
var
  i : integer;
begin
  for i := 0 to Transfers.Count - 1 do
    TTransfer (Transfers[i]).Free;
  Transfers.Free;
  TxStream.Free;
  inherited Destroy;
```

```
end;

procedure TTFTPListener.SendError (aTransfer : TTransfer; ErrCode : Word;
  ErrMsg : string);
var
  count : integer;
begin
  TxStream.Clear;
  count := 0;
  AddWord (TFTP_ERROR);
  AddWord (ErrCode);
  AddString (ErrMsg);
  SendToSocket (aTransfer.SockAddr, aTransfer.SockAddrLength, TxStream.Memory,
TxStream.Size, count);
end;

procedure TTFTPListener.SendError (aServer : TWinsock2UDPServer; ErrCode : Word;
  ErrMsg : string);
begin
  TxStream.Clear;
  AddWord (TFTP_ERROR);
  AddWord (ErrCode);
  AddString (ErrMsg);
  SendDataTo (aServer.PeerAddress, aServer.PeerPort, TxStream.Memory, TxStream.Size);
end;

procedure TTFTPListener.SendAck (aTransfer : TTransfer; BlockNo : Word);
var
  count : integer;
begin
  TxStream.Clear;
  count := 0;
  AddWord (TFTP_ACK);
  AddWord (BlockNo);
  aTransfer.BlockNo := BlockNo;
  SendToSocket (aTransfer.SockAddr, aTransfer.SockAddrLength, TxStream.Memory,
TxStream.Size, count);
end;

procedure TTFTPListener.SendDataBlock (aTransfer: TTransfer; BlockNo : Word);
var
  count : integer;
  x, l : int64;
begin
  TxStream.Clear;
  count := 0;
  AddWord (TFTP_DATA);
  AddWord (BlockNo);
  x := (int64 (BlockNo) - 1) * 512;
  if (x >= 0) and (x < aTransfer.FStream.Size) then
    begin
      aTransfer.FStream.Seek (x, soFromBeginning);
      l := aTransfer.FStream.Size - aTransfer.FStream.Position;
      if l > 512 then l := 512;
      TxStream.CopyFrom (aTransfer.FStream, l);
    end;
  aTransfer.BlockNo := BlockNo;
  SendToSocket (aTransfer.SockAddr, aTransfer.SockAddrLength, TxStream.Memory,
TxStream.Size, count);
end;

function display_string (s : string) : string;
var
  i : integer;
begin
  Result := '';
  for i := 1 to length (s) do
    if s[i] in [' '..'~'] then
      Result := Result + s[i]
    else
      Result := Result + '[' + IntToHex (ord (s[i]), 2) + ']';
end;

function ErToStr (er : Word) : string;
begin
  case er of
    erUNDEFINED                 : Result := 'Undefined';
    erFILE_NOT_FOUND            : Result := 'File not found';
    erACCESS_VIOLATION          : Result := 'Access violation';
    erALLOCATION_EXCEEDED       : Result := 'Allocation exceeded';
    erILLEGAL_OPERATION         : Result := 'Illegal Operation';
    erUNKNOWN_TRANSFER_ID       : Result := 'Unknown transfer id';
    erFILE_ALREADY_EXISTS       : Result := 'File already exists';
    erNO_SUCH_USER              : Result := 'No such user';
    erOPTION_NEGOTIATION_FAILED : Result := 'Option negotiation failed';
    else                          Result := 'Unknown ' + IntToStr (er);
  end;
end;

procedure TTFTPListener.SetOnMsg (Value : TMsgEvent);
begin
  FOnMsg := Value;
  DoMsg ('TFTP Ready.');
end;

procedure TTFTPListener.AddByte (b: byte);
begin
  TxStream.Write (b, 1);
end;

procedure TTFTPListener.AddWord (w : Word);
begin
  AddByte (w div $100);
  AddByte (w mod $100);
```

```
end;

procedure TTFTPListener.AddString (s : string);
begin
  TxStream.Write (s[1], length (s));
  AddByte (0);
end;

function TTFTPListener.GetTransfer (byID : Word): TTransfer;
var
  i : integer;
begin
  for i := 0 to Transfers.Count - 1 do
    begin
      Result := TTransfer (Transfers[i]);
      if Result.TID = byID then exit;
    end;
  Result := nil;
end;

procedure TTFTPListener.RemoveTransfer (byID : Word);
var
  aTransfer : TTransfer;
begin
  aTransfer := GetTransfer (byID);
  RemoveTransfer (aTransfer);
end;

procedure TTFTPListener.RemoveTransfer (byTransfer : TTransfer);
begin
  if byTransfer <> nil then
    begin
      Transfers.Remove (byTransfer);
      byTransfer.Free;
    end;
end;

procedure TTFTPListener.DoMsg (s: string);
begin
  if Assigned (FOnMsg) then FOnMsg (Self, s);
end;

function TTFTPListener.DoExecute (aThread : TWinsock2UDPServerThread) : Boolean;
var
  op, er, bn : Word;
  rm : int64;
  fn, mode, msg : string;
  aTransferThread : TFTPTransferThread;
  aTransfer : TTransfer;
  aFile : TFileStream;
begin
  Result := inherited DoExecute (aThread);
  if not Result then exit;
  if aThread.Server.Count > 0 then
    begin
      aTransfer := GetTransfer (aThread.Server.PeerPort);
      aTransferThread := TFTPTransferThread (aThread);
      aTransferThread.FStream.Clear;
      aTransferThread.FStream.Write (aThread.Server.Data^, aThread.Server.Count);
      aTransferThread.FStream.Seek (0, soFromBeginning);
      op := aTransferThread.ReadWord;
      case op of
        TFTP_RRQ   :
          begin
            fn := aTransferThread.ReadString;
            mode := aTransferThread.ReadString;
            DoMsg ('Transfer  for ' + fn + ' started.');
            if aTransfer = nil then
              begin
                aTransfer := TTransfer.Create (Self);
                aTransfer.TID := aThread.Server.PeerPort;
                Transfers.Add (aTransfer);
              end;
            aTransfer.SockAddrLength := 0;
            aTransfer.SockAddr := AddressToSockAddr (aThread.Server.PeerAddress,
aTransfer.SockAddrLength);
            PortToSockAddr (aThread.Server.PeerPort, aTransfer.SockAddr,
aTransfer.SockAddrLength);
            if FileExists (fn) then
              begin
                aTransfer.Op := op;
                aTransfer.FileName := fn;
                aTransfer.FStream.Clear;
                try
                  aFile := TFileStream.Create (fn, fmOpenRead);
                  aTransfer.FStream.CopyFrom (aFile, 0);
                  aFile.Free;
                  aTransfer.FStream.Seek (0, soFromBeginning);
                  SendDataBlock (aTransfer, 1)
                except
                  DoMsg ('Transfer for ' + fn + ' failed. - error opening file.');
                  SendError (aTransfer, erACCESS_VIOLATION, 'error opening ' + fn);
                end;
              end
            else
              begin
                SendError (aTransfer, erFILE_NOT_FOUND, fn);
                DoMsg ('Transfer for ' + fn + ' failed. - file for found.');
              end;
           end;
        TFTP_WRQ   :
          begin
            fn := aTransferThread.ReadString;
            mode := aTransferThread.ReadString;
            DoMsg ('Transfer for ''' + fn + ''' started.');
            if aTransfer = nil then
              begin
                aTransfer := TTransfer.Create (Self);
                aTransfer.TID := aThread.Server.PeerPort;
                Transfers.Add (aTransfer);
              end;
            aTransfer.SockAddrLength := 0;
            aTransfer.SockAddr := AddressToSockAddr (aThread.Server.PeerAddress,
aTransfer.SockAddrLength);
            PortToSockAddr (aThread.Server.PeerPort, aTransfer.SockAddr,
aTransfer.SockAddrLength);
            if mode = 'octet' then
              begin
                aTransfer.Op := op;
                aTransfer.FileName := fn;
                aTransfer.FStream.Clear;
                aTransfer.BlockNo := 0;
                SendAck (aTransfer, 0);
              end
            else
              SendError (aTransfer, erOPTION_NEGOTIATION_FAILED, mode);
          end;
        TFTP_DATA  :
          begin
            bn := aTransferThread.ReadWord;
            rm := aTransferThread.Remaining;
            if aTransfer = nil then
              SendError (aThread.Server, erUNKNOWN_TRANSFER_ID, IntToStr
(aThread.Server.PeerPort))
            else
              begin
                aTransfer.BlockNo := bn;
                aTransfer.FStream.CopyFrom (aTransferThread.FStream, rm);
                SendAck (aTransfer, bn);
                if rm < 512 then
                  begin
                    aTransfer.FStream.Seek (0, soFromBeginning);
                    DoMsg ('Transfer for ' + aTransfer.FileName + ' complete.');
                    if FileExists (aTransfer.FileName) then  DeleteFile (aTransfer.FileName);
                    try
                      aFile := TFileStream.Create (aTransfer.FileName, fmCreate);
                      aTransfer.FStream.Seek (0, soFromBeginning);
                      aFile.CopyFrom (aTransfer.FStream, aTransfer.FStream.Size);
                      aFile.Free;
                      if (aTransfer.FileName = 'kernel7.img') or (aTransfer.FileName = 'kernel.img') then
                        begin
                          DoMsg ('Restarting.');
                          SystemRestart (0);
                        end;
                    except
                      SendError (aTransfer, erACCESS_VIOLATION, 'error creating ' +
aTransfer.FileName);
                  end;
                RemoveTransfer (aTransfer);
              end;
          end;
        end;
      TFTP_ACK   :
        begin
          bn := aTransferThread.ReadWord;
          if aTransfer = nil then
            SendError (aThread.Server, erUNKNOWN_TRANSFER_ID, IntToStr
(aThread.Server.PeerPort))
          else
            begin
              if (int64 (bn) * 512) >= aTransfer.FStream.Size then
                begin
                  DoMsg ('Transfer for ' + aTransfer.FileName + ' complete.');
                  RemoveTransfer (aTransfer); // maybe should dally
                end
              else
                SendDataBlock (aTransfer, bn  + 1);
            end;
        end;
      TFTP_ERROR :
        begin
          er := aTransferThread.ReadWord;
          msg := aTransferThread.ReadString;
          DoMsg ('Error ' + ErToStr (er) + ' - ' + msg);
          RemoveTransfer (aTransfer);
        end
      else
        SendError (aThread.Server, erILLEGAL_OPERATION, IntToStr (op));  // unknown opcode
    end;
  end;
end;

procedure TTFTPListener.DoCreateThread (aServer: TWinsock2UDPServer;
  var aThread: TWinsock2UDPServerThread);
begin
  aThread := TFTPTransferThread.Create (aServer);
end;

procedure TFTPStart;
var
  WSAData : TWSAData;
begin
  WSADATA.wVersion := 0;   // prevent not initialsed warning
  FillChar (WSAData, SizeOf (TWSAData), 0);
  if WSAStartup (WINSOCK_VERSION, WSAData) = ERROR_SUCCESS then
    begin
      TFTP := TTFTPListener.Create;
      TFTP.Active := true;
    end;
end;


initialization

TFTPStart;

end.
```

Appendix B

```
program TFTPTest;

{$mode objfpc}{$H+}

uses
  RaspberryPi3,
  GlobalConfig,
  GlobalConst,
  GlobalTypes,
  Platform,
  Threads,
  SysUtils,
  Classes,
  Ultibo,
  Console,
  Winsock2,
  Shell,
  uTFTP;

var
  IPAddress : string;
  WindowHandle : TWindowHandle;

function WaitForIPComplete : string;
var
  TCP : TWinsock2TCPClient;
begin
  TCP := TWinsock2TCPClient.Create;
  Result := TCP.LocalAddress;
  if (Result = '') or (Result = '0.0.0.0') or (Result = '255.255.255.255') then
    begin
      while (Result = '') or (Result = '0.0.0.0') or (Result = '255.255.255.255') do
        begin
          sleep (1000);
          Result := TCP.LocalAddress;
        end;
    end;
  TCP.Free;
end;
```

```
procedure Msg (Sender : TObject; s : string);
begin
  ConsoleWindowWriteLn (WindowHandle, s);
end;

procedure WaitForSDDrive;
begin
  while not DirectoryExists ('C:\') do sleep (500);
end;

begin
   // open console window
  WindowHandle := ConsoleWindowCreate (ConsoleDeviceGetDefault,
CONSOLE_POSITION_FULL, false);
  ConsoleWindowWriteLn (WindowHandle, 'TFTP Demo.');
  // wait for IP address and SD Card to be initialised.
  WaitForSDDrive;
  IPAddress := WaitForIPComplete;
  ConsoleWindowWriteLn (WindowHandle, 'Local Address ' + IPAddress);
  SetOnMsg (@Msg);
  ThreadHalt (0);
end.
```