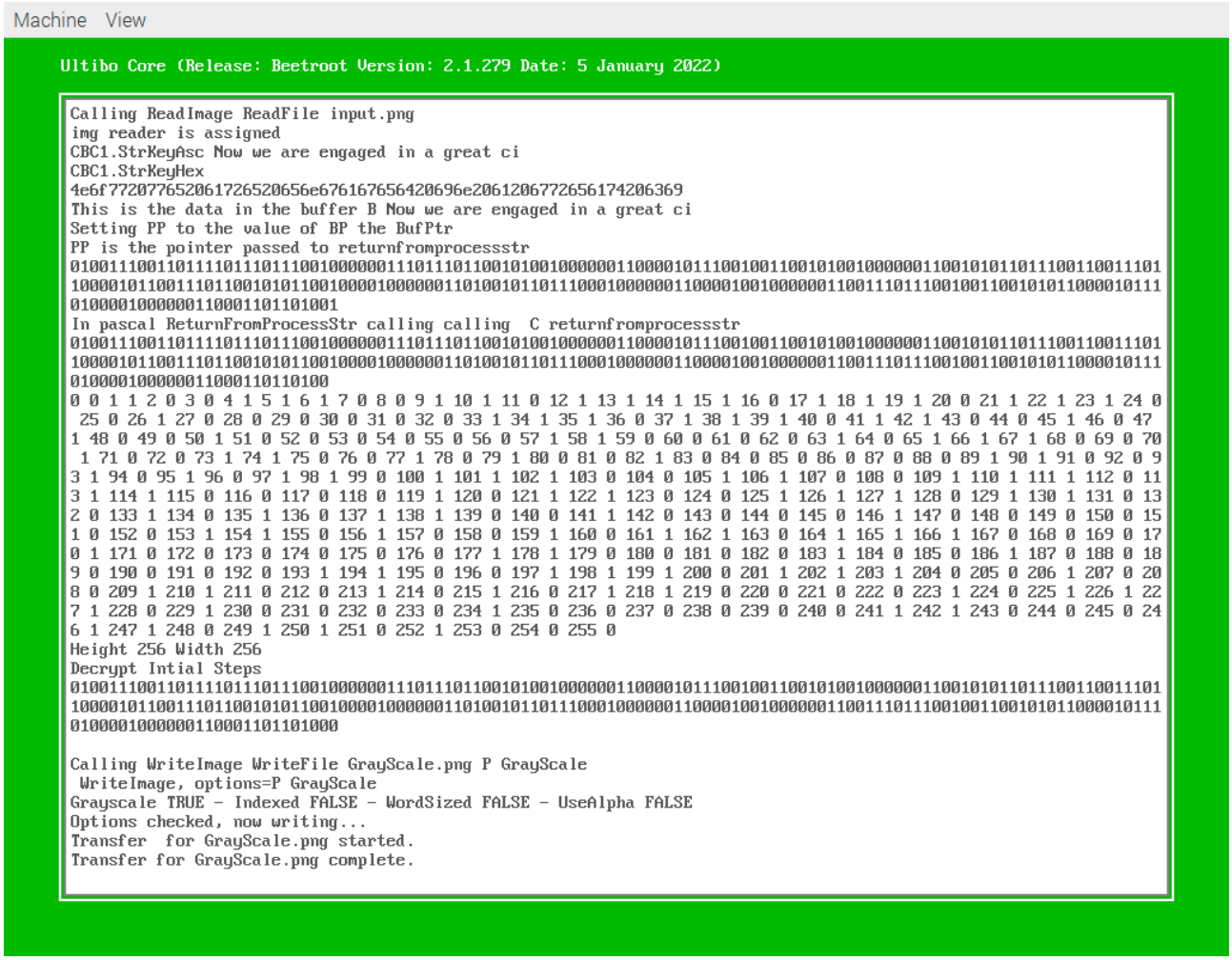
**********Draft**********

Steganography for Ultibo Bare Metal with Debug using QEMU adding Cryptography Using Crypto & APICrypto from the Ultitbo RTL

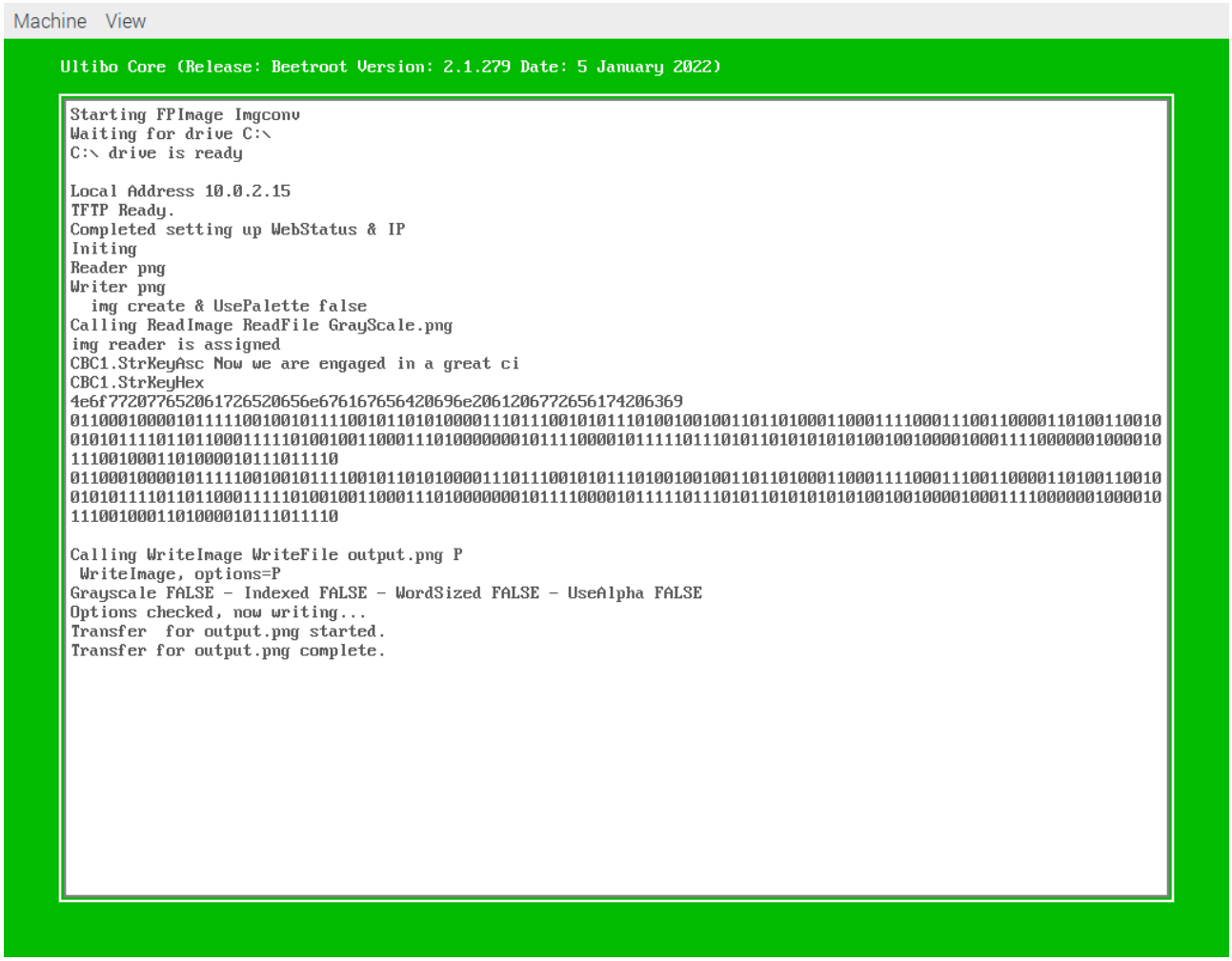
02/17/22

**********Draft**********

imgprocess1.lpr reads input.png writes GrayScale.png



imgprocess2.lpr reads GrayScale.png writes output.png



Several arrays are needed to perform the next phase of the process.

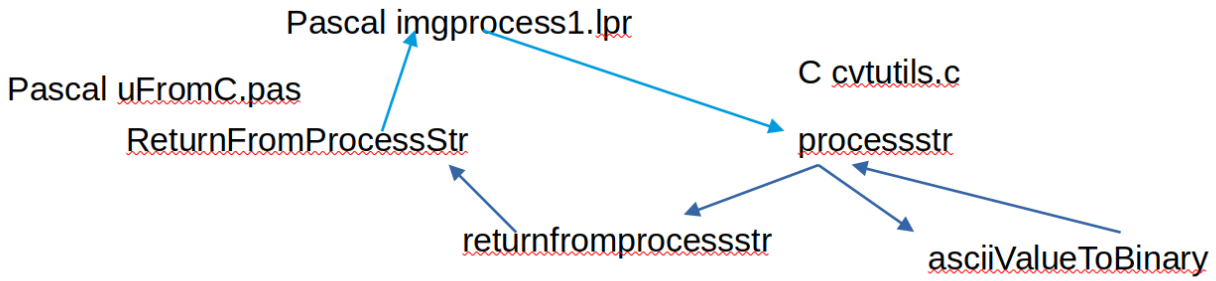
```
type
  MODR = array[0..255,0..255] of word;
  MODRPtr = ^MODR;
  XORR = array[0..255,0..255] of word;
  XORRPtr = ^XORR;
  TLSB = array[0..255,0..255] of word;
  TLSBPtr = ^TLSB;
  Lsb = array[0..255] of byte;
  lsbPtr = ^Lsb;
  Buffer = String[255];
  BufPtr = ^Buffer;   ;


S1,S2:String;
 xx,yy : LongWord;
 databuffer :PChar;
 B  : Buffer;
 BP : BufPtr;
 PP : Pointer;
 bb : Lsb;
 bbp : LsbPtr;
```

```
 modbuf : MODR;
 xorbuf : XORR;
 tlsbbuf : TLSB;
```

```
modbuf[i,j] := clr.red mod 2;

xx := modbuf[i,j] xor bbp^[i];
clr.red:=clr.red+xx;
After a 0 or 1 is added to red pixel
clr.green:=clr.red;
clr.blue:=clr.red;
```



Each char of the string is going to be place in a byte array as 0 or 1.

```
01001110011011110111011100100000011101110110010100100000011000010111001001100101
00100000011001010110111001100111011000010110011101100101011001000010000001101001
01101110001000000110000100100000011001110111001001100101011000010111010000100000
0110001101101001
```

```
01001110011011110111011100100000011101110110010100100000011000010111001001100101
10000101100111011001010110010000100000011010010110111000100000011000010010000001100111011100100110010101100000101110100001000000110000110110100
0 0 1 1 2 0 3 0 4 1 5 1 6 1 7 0 8 0 9 1 10 1 11 0 12 1 13 1 14 1 15 1 16 0 17 1 18 1 19 1 20 0 21 1 22 1 23 1 24 0
 25 0 26 1 27 0 28 0 29 0 30 0 31 0 32 0 33 1 34 1 35 1 36 0 37 1 38 1 39 1 40 0 41 1 42 1 43 0 44 0 45 1 46 0 47
1 48 0 49 0 50 1 51 0 52 0 53 0 54 0 55 0 56 0 57 1 58 1 59 0 60 0 61 0 62 0 63 1 64 0 65 1 66 1 67 1 68 0 69 0 70
 1 71 0 72 0 73 1 74 1 75 0 76 0 77 1 78 0 79 1 80 0 81 0 82 1 83 0 84 0 85 0 86 0 87 0 88 0 89 1 90 1 91 0 92 0 9
3 1 94 0 95 1 96 0 97 1 98 1 99 0 100 1 101 1 102 1 103 0 104 0 105 1 106 1 107 0 108 0 109 1 110 1 111 1 112 0 11
3 1 114 1 115 0 116 0 117 0 118 0 119 1 120 0 121 1 122 1 123 0 124 0 125 1 126 1 127 1 128 0 129 1 130 1 131 0 13
2 0 133 1 134 0 135 1 136 0 137 1 138 1 139 0 140 0 141 1 142 0 143 0 144 0
```

uFromC now returns the string with the call to **processstr('Now we are engaged in a great ci');**
ProcessStrResult String For 32 char passed the return string is 32 * 8 which 256 char.
xx := modbuf[i,j] xor bbp^[i];
clr.red:=clr.red+xx;
After a 0 or 1 is added to red pixel
clr.green:=clr.red;
clr.blue:=clr.red

```
Height 256 Width 256
0 0 0 0 41668 41668
0 0 41668 41668 41668
1 0 0 1 41415 41414
1 0 41415 41415 41415
2 0 1 1 40860 40859
2 0 40860 40860 40860
3 0 0 0 41564 41564
3 0 41564 41564 41564
4 0 1 0 41683 41683
4 0 41683 41683 41683
5 0 1 0 41215 41215
5 0 41215 41215 41215
6 0 1 0 40709 40709
6 0 1 0 40709 40709
6 0 40709 40709 40709
7 0 1 1 40174 40173
7 0 40174 40174 40174
8 0 1 1 40012 40011
8 0 40012 40012 40012
9 0 1 0 41485 41485
9 0 41485 41485 41485
10 0 0 1 39553 39552
10 0 39553 39553 39553
11 0 0 0 39898 39898
11 0 39898 39898 39898
12 0 0 1 39995 39994
12 0 39995 39995 39995
13 0 1 0 39331 39331
13 0 39331 39331 39331
```

Current Issues:
1. Conversion of RGB to gray scale using fcl-image fpimage.pp
This issue can be resolved by adding to WriteOptions := 'P GrayScale';
. This requires writing to the disk.img.
10-2-22 16:34:52 57612 GrayScale.png

```
clr.red:=round(clr.red*0.29900);
clr.blue:=round(clr.blue*0.11400);
clr.green:=round(clr.green*0.58700);
clr.green:=clr.red+clr.blue+clr.green;
clr.red:=clr.green;
clr.blue:=clr.green;
```

11-2-22 12:31:41 73068 GrayScale.png
This makes red, blue, and green all the same value. Which is what WriteOptions := 'P GrayScale';
did.

Note: The size of GrayScale.png is 26.8%

2 Need to determine how to return the results of calling processstr(S1); back to improcessing1.

Ultibo has provided some ideas on this I just do not under the steps.

This project Goal: To learn steganogrphy based on code
https://github.com/TheAlgorithms/MATLAB-Octave/blob/master/algorithms/ImageProcessing/LSB%20based%20Image%20Steganography/steganography.m
The file steganography.m RPi4B Octave only works by commenting some lines
and creating bit string of the of the desired text to embed in the image.

This repo git@github.com:develone/MATLAB-Octave.git
  which was forked from https://github.com/TheAlgorithms/MATLAB-Octave
  required minor modification to run on Raspberry Pi 4B 8Gb.

The following C program was written:

```
#include <stdio.h>
#include <string.h>

int asciiValueToBinary(int asciiInput)
{
	int res = 0, i = 1, rem;

	while (asciiInput > 0)
	{
		rem = asciiInput % 2;
		res = res + (i * rem);
		asciiInput = asciiInput / 2;
		i = i * 10;
	}
	//printf("%x\n",res);
	return(res);
}

void processstr(char  *x) {
int i,l;
l=strlen(x);
int outstr[l];
//printf("C %d %s\n",l,x);
for(i=0;i<l;i++) {
	printf("%d %08d ",i,asciiValueToBinary(*x));
  //printf("%08d",asciiValueToBinary(*x));
  outstr[i]=asciiValueToBinary(*x);
	x++;
}
printf("\n");
for(i=0;i<l;i++) printf("%08d",outstr[i]);
printf("\n");
}

int main() {
	char *p;
	char a[]="Now we are engaged in a great ci";
	p = a;
	processstr(p);
return (0);
}
```

The methods void processstr(char *x) & int asciiValueToBinary(int asciiInput) are found in a program cvtutils.c. The is compiled for usewith Ultibo using
./libuild.sh in Ultibo_Projects/imgconv/QEMU

gcc bitstring.c -o bitstring

./bitstring

0 01001110 1 01101111 2 01110111 3 00100000 4 01110111 5 01100101 6 00100000 7 01100001 8 01110010 9 01100101 10 00100000 11 01100101 12 01101110 13 01100111 14 01100001 15 01100111 16 01100101 17 01100100 18 00100000 19 01101001 20 01101110 21 00100000 22 01100001 23 00100000 24 01100111 25 01110010 26 01100101 27 01100001 28 01110100 29 00100000 30 01100011 31 01101001
01001110011011110111011100100000011101110110010100100000011000010111001001100101001000000110010101101110011001110110000101100111011001010110010000100000011010010110111000100000011000010010000001100111011100100110010101100001011101000010000001100001101101001

This information is part of readme.md provided in the original repo.
The encoding is done using the following steps:

1. Convert the image to greyscale
2. Resize the image if needed
3. Convert the message to its binary format
4. Initialize output image same as input image
5. Traverse through each pixel of the image and do the following:
   - Convert the pixel value to binary
   - Get the next bit of the message to be embedded
   - Create a variable temp
   If the message bit and the LSB of the pixel are same, set temp = 0
   If the message bit and the LSB of the pixel are different, set temp = 1
   This setting of temp can be done by taking XOR of message bit and the LSB of the pixel
   Update the pixel of output image to input image pixel value + temp

   Keep updating the output image till all the bits in the message are embedded
   Finally, write the input as well as the output image to local system.

The decoding/decryption is done using the following steps:

1. Get the output image which was encoded earlier.
2. Input the length of the encoded message (character count).
3. Retrieve the LSBs of each pixel
4. Form a bit sequence from these LSBs
5. Arrange the bit sequence into a matrix of 8 rows and total_message_bits/8 columns
   (each column will represent a character of 8 bits, hence 8 rows)
   - Convert the binary value to decimal
   - Get the corresponding char from ascii

   Finally, display the original message.

Now we are engaged in a great ci

octave
GNU Octave, version 6.2.0
Copyright (C) 2021 The Octave Project Developers.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for "arm-unknown-linux-gnueabihf".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1>steganograpyhy


```
1989.00,  173.346
```

octave:2> decrypt
Enter the length (character count) of the message you are looking for:
Enter the length (character count) of the message you are looking for: 32
The original message is: Now we are engaged in a great ci
octave:3>

Testing using lena_rgb_256.png in input.png

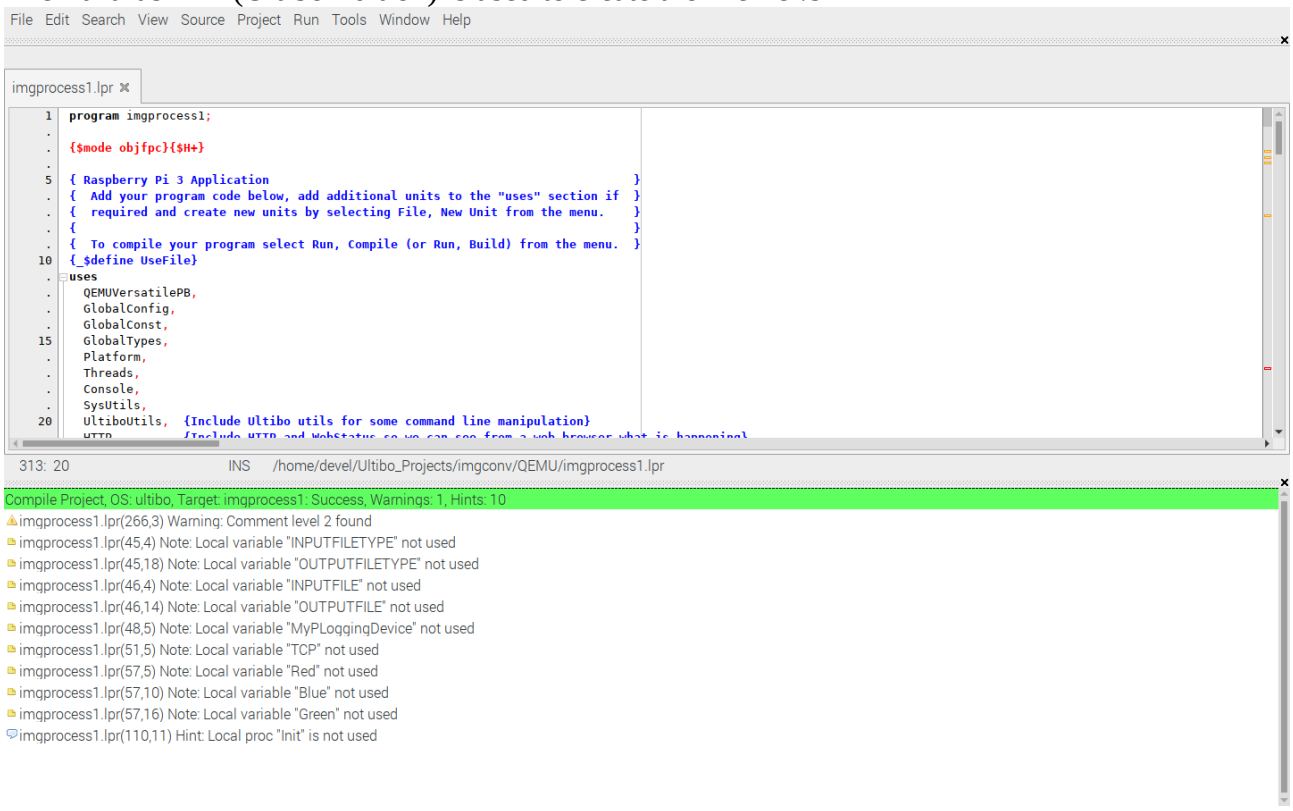
octave:1>steganograpyhy

325.825, -27.3639

octave:2> decrypt
Enter the length (character count) of the message you are looking for:
Enter the length (character count) of the message you are looking for: 32

The original message is: Now we are engaged in a great ci
octave:3>

The Lazarus IDE (Ultibo Edition) is used to create the "kernel.bin"



From the main menu Run/Compile. If no errors a green bar appers.

. Ultibo_Projects/picoultibo.sh This sets the PATH

/home/devel/ultibo/core:/home/devel/qemu-6.1.0-rpios/bin:/home/devel/local/openocd/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games

./libuild.sh in Ultibo_Projects/imgconv/QEMU

cd Ultibo_Projects/imgconv/QEMU

~/Ultibo_Projects/imgconv/QEMU $ ./startqemu.sh

While QEMU is running telnet, tftp and a webserver are provided.

~/Ultibo_Projects/imgconv/QEMU $ telnet xx.xx.xx.xx 5023
(Type HELP for a list of available commands)
>dir
Directory of C:\

28-7-21	18:41:54	53	Another File.txt
28-7-21	18:41:54	31	Test File.txt
28-7-21	18:41:54	<DIR>	www
10-1-22	12:25:18	24	testfile
10-1-22	23:54:30	24	junk

```
10-1-22 12:25:18              24  256com
5-2-22 17:32:46            65536  red
28-7-21 18:44:28              24  256decom
28-7-21 18:44:28           196730  lena_rgb_256.bmp
28-7-21 18:44:28           196730  MyBitmap.bmp
5-2-22 17:41:56              7848  test.j2k
5-2-22 17:32:48            65536  green
5-2-22 17:32:48            65536  blue
5-2-22 17:32:48           196730  test_wr.bmp
5-2-22 17:57:08           125663  lena_rgb_256.png
7-2-22 12:54:36           196662  lena_rgb_256_fpng.bmp
       15 file(s) 1117151 bytes
       1 dir(s)

C:\>logout
Goodbye!
Connection closed by foreign host.

~/Ultibo_Projects/imgconv/QEMU $ tftp xx.xx.xx.xx 5069
tftp> binary
tftp> get lena_rgb_256_fpng.bmp
Received 196662 bytes in 2.8 seconds
tftp> quit
```

http://xx.xx.xx.xx:5080/status

xxx