# TFTP

Generated by Doxygen 1.8.14

# Contents

## 1 Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

## 2 File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

# 3 Data Structure Documentation

## 3.1 fblock Struct Reference

Structure which defines a file.

```
#include <fblock.h>
```

#### Data Fields

- FILE ∗ file

  *Pointer to the file.*
- int block_size

  *Predefined block size for i/o operations.*
- char mode

  *Can be read xor write, text xor binary.*
- union {
    int written
      *Bytes already written (for future use)*
    int remaining
      *Remaining bytes to read.*
  };

### 3.1.1 Detailed Description

Structure which defines a file.

Definition at line 31 of file fblock.h.

### 3.1.2 Field Documentation

#### 3.1.2.1 mode

```
char fblock::mode
```

Can be read xor write, text xor binary.

Definition at line 34 of file fblock.h.

The documentation for this struct was generated from the following file:

- fblock.h

# 4 File Documentation

## 4.1 fblock.c File Reference

Implementation of fblock.h .

```
#include "include/fblock.h"
#include <stdio.h>
#include <string.h>
#include "include/logging.h"
```

#### Macros

- #define **LOG_LEVEL** LOG_INFO

#### Functions

- int get_length (FILE ∗f)

    *Returns file length.*
- struct fblock fblock_open (char ∗filename, int block_size, char mode)

    *Opens a file.*
- int fblock_read (struct fblock ∗m_fblock, char ∗buffer)

    *Reads next block_size bytes from file.*
- int fblock_write (struct fblock ∗m_fblock, char ∗buffer, int block_size)

    *Writes next block_size bytes to file.*
- int **fblock_close** (struct fblock ∗m_fblock)

### 4.1.1 Detailed Description

Implementation of fblock.h .

#### Author

Riccardo Mancini

Definition in file fblock.c.

### 4.1.2 Function Documentation

#### 4.1.2.1 fblock_open()

```
struct fblock fblock_open (
            char * filename,
            int block_size,
            char mode )
```

Opens a file.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *block_size* | size of the blocks |
| *modern* | mode (read, write, text, binary) |

**Returns**

fblock structure

**See also**

FBLOCK_MODE_TEXT
FBLOCK_MODE_BINARY
FBLOCK_WRITE
FBLOCK_READ

Definition at line 29 of file fblock.c.

#### 4.1.2.2 fblock_read()

```
int fblock_read (
             struct fblock * m_fblock,
             char * buffer )
```

Reads next block_size bytes from file.

**Parameters**

| | |
|---|---|
| *m_fblock* | fblock instance |
| *buffer* | block_size bytes buffer |

**Returns**

0 in case of success, otherwise number of bytes it could not read

Definition at line 67 of file fblock.c.

#### 4.1.2.3 fblock_write()

```
int fblock_write (
             struct fblock * m_fblock,
             char * buffer,
             int block_size )
```

Writes next block_size bytes to file.

**Parameters**

| | |
|---|---|
| *m_fblock* | fblock instance |
| *buffer* | block_size bytes buffer |
| *block_size* | if set to a non-0 value, override block_size defined in fblock. |

**Returns**

0 in case of success, otherwise number of bytes it could not write

Definition at line 82 of file fblock.c.

#### 4.1.2.4 get_length()

```
int get_length (
              FILE * f )
```

Returns file length.

**Parameters**

| | |
|---|---|
| *f* | file pointer |

**Returns**

file length in bytes

Definition at line 20 of file fblock.c.

## 4.2 fblock.c

```
00001
00007 #define LOG_LEVEL LOG_INFO
00008
00009 #include "include/fblock.h"
00010 #include <stdio.h>
00011 #include <string.h>
00012 #include "include/logging.h"
00013
00020 int get_length(FILE *f){
00021   int size;
00022   fseek(f, 0, SEEK_END); // seek to end of file
00023   size = ftell(f); // get current file pointer
00024   fseek(f, 0, SEEK_SET); // seek back to beginning of file
00025   return size;
00026 }
00027
00028
00029 struct fblock fblock_open(char* filename, int block_size, char
      mode){
00030   struct fblock m_fblock;
00031   m_fblock.block_size = block_size;
00032   m_fblock.mode = mode;
00033
00034   char mode_str[4] = "";
00035
00036   LOG(LOG_DEBUG, "Opening file %s (%s %s), block_size = %d",
00037       filename,
      (mode & FBLOCK_MODE_MASK) == FBLOCK_MODE_BINARY ? "binary" : "text",
      (mode & FBLOCK_RW_MASK) == FBLOCK_WRITE ? "write" : "read",
      block_size
  );

  if ((mode & FBLOCK_RW_MASK) == FBLOCK_WRITE){
    strcat(mode_str, "w");
    m_fblock.written = 0;
  } else {
    strcat(mode_str, "r");
  }

  if ((mode & FBLOCK_MODE_MASK) == FBLOCK_MODE_BINARY)
    strcat(mode_str, "b");
  // text otherwise

  m_fblock.file = fopen(filename, mode_str);
  if (m_fblock.file == NULL){
    LOG(LOG_ERR, "Error while opening file %s", filename);
    return m_fblock;
  }
  if ((mode & FBLOCK_RW_MASK) == FBLOCK_READ)
    m_fblock.remaining = get_length(m_fblock.file);

  LOG(LOG_DEBUG, "Successfully opened file");
  return m_fblock;
}


int fblock_read(struct fblock *m_fblock, char* buffer){
  int bytes_read, bytes_to_read;

  if (m_fblock->remaining > m_fblock->block_size)
    bytes_to_read = m_fblock->block_size;
  else
    bytes_to_read = m_fblock->remaining;

  bytes_read = fread(buffer, sizeof(char), bytes_to_read, m_fblock->file);
  m_fblock->remaining -= bytes_read;

  return bytes_to_read - bytes_read;
}


int fblock_write(struct fblock *m_fblock, char* buffer, int block_size){
  int written_bytes;

  if (!block_size)
    block_size = m_fblock->block_size;

  written_bytes = fwrite(buffer, sizeof(char), block_size, m_fblock->file);
  m_fblock->written += written_bytes;
  return block_size - written_bytes;
}

int fblock_close(struct fblock *m_fblock){
  return fclose(m_fblock->file);
}
```

## 4.3 fblock.h File Reference

File block read and write.

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

### Data Structures

- struct fblock

  *Structure which defines a file.*

### Macros

- #define FBLOCK_MODE_MASK 0b01

  *Mask for getting text/binary mode.*
- #define FBLOCK_MODE_TEXT 0b00

  *Open file in text mode.*
- #define FBLOCK_MODE_BINARY 0b01

  *Open file in binary mode.*
- #define FBLOCK_RW_MASK 0b10

  *Mask for getting r/w mode.*
- #define FBLOCK_READ 0b00

  *Open file in read mode.*
- #define FBLOCK_WRITE 0b10

  *Open file in write mode.*

### Functions

- struct fblock fblock_open (char ∗filename, int block_size, char mode)

  *Opens a file.*
- int fblock_read (struct fblock ∗m_fblock, char ∗buffer)

  *Reads next block_size bytes from file.*
- int fblock_write (struct fblock ∗m_fblock, char ∗buffer, int block_size)

  *Writes next block_size bytes to file.*
- int **fblock_close** (struct fblock ∗m_fblock)

### 4.3.1 Detailed Description

File block read and write.

**Author**

Riccardo Mancini This library provides functions for reading and writing a text or binary file using a predefined block size.

Definition in file fblock.h.

### 4.3.2 Function Documentation

#### 4.3.2.1 fblock_open()

```
struct fblock fblock_open (
            char * filename,
            int block_size,
            char mode )
```

Opens a file.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *block_size* | size of the blocks |
| *modern* | mode (read, write, text, binary) |

**Returns**

fblock structure

**See also**

FBLOCK_MODE_TEXT
FBLOCK_MODE_BINARY
FBLOCK_WRITE
FBLOCK_READ

Definition at line 29 of file fblock.c.

#### 4.3.2.2 fblock_read()

```
int fblock_read (
              struct fblock * m_fblock,
              char * buffer )
```

Reads next block_size bytes from file.

**Parameters**

| | |
|---|---|
| *m_fblock* | fblock instance |
| *buffer* | block_size bytes buffer |

**Returns**

0 in case of success, otherwise number of bytes it could not read

Definition at line 67 of file fblock.c.

#### 4.3.2.3 fblock_write()

```
int fblock_write (
              struct fblock * m_fblock,
              char * buffer,
              int block_size )
```

Writes next block_size bytes to file.

**Parameters**

| | |
|---|---|
| *m_fblock* | fblock instance |
| *buffer* | block_size bytes buffer |
| *block_size* | if set to a non-0 value, override block_size defined in fblock. |

**Returns**

0 in case of success, otherwise number of bytes it could not write

Definition at line 82 of file fblock.c.

## 4.4 fblock.h

```
#ifndef FBLOCK
#define FBLOCK

#include <stdio.h>

#define FBLOCK_MODE_MASK   0b01

#define FBLOCK_MODE_TEXT   0b00

#define FBLOCK_MODE_BINARY 0b01

#define FBLOCK_RW_MASK     0b10

#define FBLOCK_READ        0b00

#define FBLOCK_WRITE       0b10


struct fblock{
  FILE *file;
  int block_size;
  char mode;
  union{
    int written;
    int remaining;
  };
};

struct fblock fblock_open(char* filename, int block_size, char mode);

int fblock_read(struct fblock *m_fblock, char* buffer);

int fblock_write(struct fblock *m_fblock, char* buffer, int block_size);

int fblock_close(struct fblock *m_fblock);

#endif
```

## 4.5 tftp_client.c File Reference

Implementation of the TFTP client making only read requests.

```
#include "include/logging.h"
#include "include/tftp_msgs.h"
#include "include/tftp.h"
#include "include/fblock.h"
#include "include/inet_utils.h"
#include "include/debug_utils.h"
#include <arpa/inet.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

#### Macros

- #define **LOG_LEVEL** LOG_INFO
- #define READ_BUFFER_SIZE 80

  *max stdin line length*
- #define **MAX_ARGS** 3
- #define **MODE_TXT** "txt"
- #define **MODE_BIN** "bin"

#### Functions

- void split_string (char ∗line, char ∗delim, int max_argc, int ∗argc, char ∗∗argv)

  *Splits a string at each delim.*
- void print_help ()

  *Prints command usage information.*
- void cmd_help ()

  *Handles !help command, printing information about available commands.*
- void cmd_mode (char ∗new_mode)

  *Handles !mode command, changing mode to either bin or text.*
- int cmd_get (char ∗remote_filename, char ∗local_filename, char ∗sv_ip, int sv_port)

  *Handles !get command, reading file from server.*
- void cmd_quit ()

  *Handles !quit command.*
- int **main** (int argc, char ∗∗argv)

#### Variables

- char ∗ **transfer_mode**

### 4.5.1 Detailed Description

Implementation of the TFTP client making only read requests.

#### Author

Riccardo Mancini

Definition in file tftp_client.c.

### 4.5.2 Function Documentation

#### 4.5.2.1 split_string()

```
void split_string (
              char * line,
              char * delim,
              int max_argc,
              int * argc,
              char ** argv )
```

Splits a string at each delim.

Trailing LF will be removed. Consecutive delimiters will be considered as one.

**Parameters**

| | |
|---|---|
| *line* | the string to split |
| *delim* | the delimiter |
| *max_argc* | maximum number of parts to split the line into |
| *argc* | parts count (out) |
| *argv* | array of parts (out) |

Definition at line 45 of file tftp_client.c.

## 4.6 tftp_client.c

```
00001
00007 #define LOG_LEVEL LOG_INFO
00008
00009 #include "include/logging.h"
00010 #include "include/tftp_msgs.h"
00011 #include "include/tftp.h"
00012 #include "include/fblock.h"
00013 #include "include/inet_utils.h"
00014 #include "include/debug_utils.h"
00015 #include <arpa/inet.h>
00016 #include <sys/types.h>
00017 #include <sys/socket.h>
00018 #include <netinet/in.h>
00019 #include <string.h>
00020 #include <stdio.h>
00021 #include <stdlib.h>
00022 #include <time.h>
00023
00025 #define READ_BUFFER_SIZE 80
00026
00027 #define MAX_ARGS 3
00028
00029 #define MODE_TXT "txt"
00030 #define MODE_BIN "bin"
00031
00032 char* transfer_mode;
00033
00045 void split_string(char* line, char* delim, int max_argc, int *argc, char **argv){
00046   char *ptr;
00047   int len;
00048   char *pos;
00049
00050   // remove trailing LF
00051   if ((pos=strchr(line, '\n')) != NULL)
00052     *pos = '\0';
00053
```

```c
  *argc = 0;


  ptr = strtok(line, delim);

  while(ptr != NULL && *argc <= max_argc){
    len = strlen(ptr);
    if (len == 0)
      continue;

    LOG(LOG_DEBUG, "arg[%d] = '%s'", *argc, ptr);

    argv[*argc] = malloc(strlen(ptr)+1);
    strcpy(argv[*argc], ptr);

    ptr = strtok(NULL, delim);
    (*argc)++;
  }
}

void print_help(){
  printf("Usage: ./tftp_client SERVER_IP SERVER_PORT\n");
  printf("Example: ./tftp_client 127.0.0.1 69");
}


void cmd_help(){
  printf("Sono disponibili i seguenti comandi:\n");
  printf("!help --> mostra l'elenco dei comandi disponibili\n");
  printf("!mode {txt|bin} --> imposta il modo di trasferimento dei file (testo o binario)\n");
  printf("!get filename nome_locale --> richiede al server il nome del file <filename> e lo salva localmente con il nome <nome_locale>\n");
  printf("!quit --> termina il client\n");
}


void cmd_mode(char* new_mode){
  if (strcmp(new_mode, MODE_TXT) == 0){
    transfer_mode = TFTP_STR_NETASCII;
    printf("Modo di trasferimento testo configurato\n");
  } else if (strcmp(new_mode, MODE_BIN) == 0){
    transfer_mode = TFTP_STR_OCTET;
    printf("Modo di trasferimento binario configurato\n");
  } else{
    printf("Modo di traferimento sconosciuto: %s. Modi disponibili: txt, bin\n", new_mode);
  }
}


int cmd_get(char* remote_filename, char* local_filename, char* sv_ip, int sv_port){
  struct sockaddr_in my_addr, sv_addr;
  int sd;
  int ret, tid;
  struct fblock m_fblock;

  LOG(LOG_INFO, "Initializing...\n");

  sd = socket(AF_INET, SOCK_DGRAM, 0);
  if (strcmp(transfer_mode, TFTP_STR_OCTET) == 0)
    m_fblock = fblock_open(local_filename, TFTP_DATA_BLOCK, FBLOCK_WRITE|FBLOCK_MODE_BINARY);
  else if (strcmp(transfer_mode, TFTP_STR_NETASCII) == 0)
    m_fblock = fblock_open(local_filename, TFTP_DATA_BLOCK, FBLOCK_WRITE|FBLOCK_MODE_TEXT);
  else
    return 2;

  LOG(LOG_INFO, "Opening socket...");

  sv_addr = make_sv_sockaddr_in(sv_ip, sv_port);
  my_addr = make_my_sockaddr_in(0);
  tid = bind_random_port(sd, &my_addr);
  if (tid == 0){
    LOG(LOG_ERR, "Error while binding to random port");
    perror("Could not bind to random port:");
    fblock_close(&m_fblock);
    return 1;
  } else
    LOG(LOG_INFO, "Bound to port %d", tid);

  printf("Richiesta file %s (%s) al server in corso.\n", remote_filename, transfer_mode);

  ret = tftp_send_rrq(remote_filename, transfer_mode, sd, &sv_addr);
  if (ret != 0){
    fblock_close(&m_fblock);
    return 8+ret;
  }

  printf("Trasferimento file in corso.\n");

  ret = tftp_receive_file(&m_fblock, sd, &sv_addr);

  if (ret == 1){     // File not found
    printf("File non trovato.\n");
    fblock_close(&m_fblock);
    return 0;
  } else if (ret != 0){
    LOG(LOG_ERR, "Error while receiving file!");
    fblock_close(&m_fblock);
    return 16+ret;
  } else{
    int n_blocks = m_fblock.written/m_fblock.block_size + 1;
    printf("Trasferimento completato (%d/%d blocchi)\n", n_blocks, n_blocks);
    printf("Salvataggio %s completato.\n", local_filename);
    fblock_close(&m_fblock);
    return 0;
  }

}

void cmd_quit(){
  printf("Client terminato con successo\n");
  exit(0);
}

int main(int argc, char** argv){
  char* sv_ip;
  short int sv_port;
  int ret;
  char *read_buffer;
  int cmd_argc;
  char *cmd_argv[MAX_ARGS];

  //init random seed
  srand(time(NULL));

  // default mode = bin
  transfer_mode = TFTP_STR_OCTET;

  read_buffer = malloc(READ_BUFFER_SIZE);

  if (argc != 3){
    print_help();
    return 1;
  }

  // TODO: check args
  sv_ip = argv[1];
  sv_port = atoi(argv[2]);

  while(1){
    printf("> ");
    fflush(stdout); // flush stdout buffer
    fgets(read_buffer, READ_BUFFER_SIZE, stdin);
    split_string(read_buffer, " ", MAX_ARGS, &cmd_argc, cmd_argv);

    if (cmd_argc == 0){
      printf("Comando non riconosciuto : ''\n");
      cmd_help();
    } else{
      if (strcmp(cmd_argv[0], "!mode") == 0){
        if (cmd_argc == 2)
          cmd_mode(cmd_argv[1]);
        else
          printf("Il comando richiede un solo argomento: bin o txt\n");
      } else if (strcmp(cmd_argv[0], "!get") == 0){
        if (cmd_argc == 3){
          ret = cmd_get(cmd_argv[1], cmd_argv[2], sv_ip, sv_port);
          LOG(LOG_INFO, "cmd_get returned value: %d", ret);
        } else{
           printf("Il comando richiede due argomenti: <filename> e <nome_locale>\n");
        }
      } else if (strcmp(cmd_argv[0], "!quit") == 0){
        if (cmd_argc == 1){
          cmd_quit();
        } else{
           printf("Il comando non richiede argomenti\n");
        }
      } else if (strcmp(cmd_argv[0], "!help") == 0){
        if (cmd_argc == 1){
          cmd_help();
        } else{
           printf("Il comando non richiede argomenti\n");
        }
      } else {
        printf("Comando non riconosciuto : '%s'\n", cmd_argv[0]);
        cmd_help();
      }
    }
  }

  return ret;
}
```

## 4.7 tftp_msgs.c File Reference

Implementation of tftp_msgs.h .

```
#include "include/tftp_msgs.h"
#include <string.h>
#include <stdio.h>
#include "include/logging.h"
```

### Macros

- #define **LOG_LEVEL** LOG_INFO

### Functions

- int tftp_msg_type (char ∗buffer)

  *Retuns msg type given message buffer.*
- void tftp_msg_build_rrq (char ∗filename, char ∗mode, char ∗buffer)

  *Builds a read request message.*
- int tftp_msg_unpack_rrq (char ∗buffer, int buffer_len, char ∗filename, char ∗mode)

  *Unpacks a read request message.*
- int tftp_msg_get_size_rrq (char ∗filename, char ∗mode)

  *Returns size in bytes of a read request message.*
- void tftp_msg_build_wrq (char ∗filename, char ∗mode, char ∗buffer)

  *Builds a write request message.*
- int **tftp_msg_unpack_wrq** (char ∗buffer, int buffer_len, char ∗filename, char ∗mode)
- int tftp_msg_get_size_wrq (char ∗filename, char ∗mode)

  *Returns size in bytes of a write request message.*
- void tftp_msg_build_data (int block_n, char ∗data, int data_size, char ∗buffer)

  *Builds a data message.*
- int tftp_msg_unpack_data (char ∗buffer, int buffer_len, int ∗block_n, char ∗data, int ∗data_size)

  *Unpacks a data message.*
- int tftp_msg_get_size_data (int data_size)

  *Returns size in bytes of a data message.*
- void tftp_msg_build_ack (int block_n, char ∗buffer)

  *Builds an acknowledgment message.*
- int tftp_msg_unpack_ack (char ∗buffer, int buffer_len, int ∗block_n)

  *Unpacks an acknowledgment message.*
- int tftp_msg_get_size_ack ()

  *Returns size in bytes of an acknowledgment message.*
- void tftp_msg_build_error (int error_code, char ∗error_msg, char ∗buffer)

  *Builds an error message.*
- int tftp_msg_unpack_error (char ∗buffer, int buffer_len, int ∗error_code, char ∗error_msg)

  *Unpacks an error message.*
- int tftp_msg_get_size_error (char ∗error_msg)

  *Returns size in bytes of an error message.*

### 4.7.1 Detailed Description

Implementation of tftp_msgs.h .

**Author**

Riccardo Mancini

Definition in file tftp_msgs.c.

### 4.7.2 Function Documentation

#### 4.7.2.1 tftp_msg_build_ack()

```
void tftp_msg_build_ack (
              int block_n,
              char * buffer )
```

Builds an acknowledgment message.

Message format:

```
 2 bytes    2 bytes
 -------------------
| 04     |   Block #  |
 --------------------
```

**Parameters**

| | |
|---|---|
| *block_n* | block sequence number |
| *buffer* | data buffer where to build the message |

Definition at line 144 of file tftp_msgs.c.

#### 4.7.2.2 tftp_msg_build_data()

```
void tftp_msg_build_data (
              int block_n,
              char * data,
              int data_size,
              char * buffer )
```

Builds a data message.

Message format:

```
 2 bytes    2 bytes        n bytes
 ----------------------------------
| 03     |   Block #  |    Data     |
 ----------------------------------
```

**Parameters**

| | |
|---|---|
| *block_n* | block sequence number |
| *data* | pointer to the buffer containing the data to be transfered |
| *data_size* | data buffer size |
| *buffer* | data buffer where to build the message |

Definition at line 113 of file tftp_msgs.c.

#### 4.7.2.3 tftp_msg_build_error()

```
void tftp_msg_build_error (
             int error_code,
             char * error_msg,
             char * buffer )
```

Builds an error message.

Message format:

```
  2 bytes  2 bytes        string    1 byte
 ----------------------------------------
| 05     |  ErrorCode |    ErrMsg    |   0  |
 ----------------------------------------
```

**Parameters**

| | |
|---|---|
| *error_code* | error code (from 0 to 7) |
| *error_msg* | error message |
| *buffer* | data buffer where to build the message |

Definition at line 169 of file tftp_msgs.c.

#### 4.7.2.4 tftp_msg_build_rrq()

```
void tftp_msg_build_rrq (
             char * filename,
             char * mode,
             char * buffer )
```

Builds a read request message.

**2 bytes string 1 byte string 1 byte**

**| 01 | Filename | 0 | Mode | 0 |**

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |
| *buffer* | data buffer where to build the message |

Definition at line 18 of file tftp_msgs.c.

#### 4.7.2.5 tftp_msg_build_wrq()

```
void tftp_msg_build_wrq (
             char * filename,
             char * mode,
             char * buffer )
```

Builds a write request message.

Message format:

```
 2 bytes    string    1 byte    string    1 byte
 -----------------------------------------------
|   02   |  Filename  |   0  |    Mode    |   0  |
 -----------------------------------------------
```

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |
| *buffer* | data buffer where to build the message |

Definition at line 65 of file tftp_msgs.c.

#### 4.7.2.6 tftp_msg_get_size_ack()

```
int tftp_msg_get_size_ack ( )
```

Returns size in bytes of an acknowledgment message.

It just returns 4.

**Parameters**

| | |
|---|---|
| *data_size* | data buffer size |

**Returns**

size in bytes

Definition at line 165 of file tftp_msgs.c.

#### 4.7.2.7 tftp_msg_get_size_data()

```
int tftp_msg_get_size_data (
             int data_size )
```

Returns size in bytes of a data message.

It just sums 4 to data_size.

**Parameters**

| | |
|---|---|
| *data_size* | data buffer size |

**Returns**

size in bytes

Definition at line 140 of file tftp_msgs.c.

#### 4.7.2.8 tftp_msg_get_size_error()

```
int tftp_msg_get_size_error (
             char * error_msg )
```

Returns size in bytes of an error message.

**Parameters**

| | |
|---|---|
| *error_msg* | error message |

**Returns**

size in bytes

Definition at line 204 of file tftp_msgs.c.

#### 4.7.2.9 tftp_msg_get_size_rrq()

```
int tftp_msg_get_size_rrq (
             char * filename,
             char * mode )
```

Returns size in bytes of a read request message.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |

**Returns**

size in bytes

Definition at line 61 of file tftp_msgs.c.

#### 4.7.2.10 tftp_msg_get_size_wrq()

```
int tftp_msg_get_size_wrq (
            char * filename,
            char * mode )
```

Returns size in bytes of a write request message.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |

**Returns**

size in bytes

Definition at line 109 of file tftp_msgs.c.

#### 4.7.2.11 tftp_msg_type()

```
int tftp_msg_type (
            char * buffer )
```

Retuns msg type given message buffer.

**Parameters**

| | |
|---|---|
| *buffer* | the buffer |

**Returns**

message type

**See also**

TFTP_TYPE_RRQ
TFTP_TYPE_WRQ
TFTP_TYPE_DATA
TFTP_TYPE_ACK
TFTP_TYPE_ERROR

Definition at line 14 of file tftp_msgs.c.

#### 4.7.2.12 tftp_msg_unpack_ack()

```
int tftp_msg_unpack_ack (
             char * buffer,
             int buffer_len,
             int * block_n )
```

Unpacks an acknowledgment message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *block_n* | pointer where block_n will be written [out] |
| *data* | pointer inside buffer where the data is [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 151 of file tftp_msgs.c.

#### 4.7.2.13 tftp_msg_unpack_data()

```
int tftp_msg_unpack_data (
             char * buffer,
             int buffer_len,
             int * block_n,
             char * data,
             int * data_size )
```

Unpacks a data message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *block_n* | pointer where block_n will be written [out] |
| *data* | pointer where to copy data [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 122 of file tftp_msgs.c.

#### 4.7.2.14 tftp_msg_unpack_error()

```
int tftp_msg_unpack_error (
             char * buffer,
             int buffer_len,
             int * error_code,
             char * error_msg )
```

Unpacks an error message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *error_code* | pointer where error_code will be written [out] |
| *error_msg* | pointer to error message inside the message [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 178 of file tftp_msgs.c.

#### 4.7.2.15 tftp_msg_unpack_rrq()

```
int tftp_msg_unpack_rrq (
             char * buffer,
             int buffer_len,
             char * filename,
             char * mode )
```

Unpacks a read request message.

Unpacks a write request message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *filename* | name of the file [out] |
| *mode* | requested transfer mode ("netascii" or "octet") [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 27 of file tftp_msgs.c.

## 4.8 tftp_msgs.c

```
00001
00007 #define LOG_LEVEL LOG_INFO
00008
00009 #include "include/tftp_msgs.h"
00010 #include <string.h>
00011 #include <stdio.h>
00012 #include "include/logging.h"
00013
00014 int tftp_msg_type(char *buffer){
00015   return (((int)buffer[0]) << 8) + buffer[1];
00016 }
00017
00018 void tftp_msg_build_rrq(char* filename, char* mode, char* buffer){
00019   buffer[0] = 0;
00020   buffer[1] = 1;
00021   buffer += 2;
00022   strcpy(buffer, filename);
00023   buffer += strlen(filename)+1;
00024   strcpy(buffer, mode);
00025 }
00026
00027 int tftp_msg_unpack_rrq(char* buffer, int buffer_len, char* filename, char*
      mode){
00028   int offset = 0;
00029   if (tftp_msg_type(buffer) != TFTP_TYPE_RRQ){
00030     LOG(LOG_ERR, "Expected RRQ message (1), found %d", tftp_msg_type(buffer));
00031     return 1;
00032   }
00033
00034   offset += 2;
00035   if (strlen(buffer+offset) > TFTP_MAX_FILENAME_LEN){
00036     LOG(LOG_ERR, "Filename too long (%d > %d): %s", (int) strlen(buffer+offset), TFTP_MAX_FILENAME_LEN,
      buffer+offset);
00037     return 3;
00038   }
00039   strcpy(filename, buffer+offset);
00040
00041   offset += strlen(filename)+1;
00042   if (strlen(buffer+offset) > TFTP_MAX_MODE_LEN){
00043     LOG(LOG_ERR, "Mode string too long (%d > %d): %s", (int) strlen(buffer+offset), TFTP_MAX_MODE_LEN,
      buffer+offset);
00044     return 4;
00045   }
00046   strcpy(mode, buffer+offset);
00047
00048   offset += strlen(mode)+1;
00049   if (buffer_len != offset){
00050     LOG(LOG_ERR, "Packet contains unexpected fields");
00051     return 2;
00052   }
00053   if (strcmp(mode, TFTP_STR_NETASCII) == 0 || strcmp(mode, TFTP_STR_OCTET) == 0)
00054     return 0;
00055   else{
00056     LOG(LOG_ERR, "Unrecognized transfer mode: %s", mode);
00057     return 5;
00058   }
00059 }
00060
00061 int tftp_msg_get_size_rrq(char* filename, char* mode){
00062   return 4 + strlen(filename) + strlen(mode);
00063 }
00064
00065 void tftp_msg_build_wrq(char* filename, char* mode, char* buffer){
00066   buffer[0] = 0;
00067   buffer[1] = 2;
00068   buffer += 2;
00069   strcpy(buffer, filename);
00070   buffer += strlen(filename)+1;
00071   strcpy(buffer, mode);
00072 }
00073
00074 int tftp_msg_unpack_wrq(char* buffer, int buffer_len, char* filename, char* mode){
00075   int offset = 0;
```

```
  if (tftp_msg_type(buffer) != TFTP_TYPE_WRQ){
    LOG(LOG_ERR, "Expected WRQ message (2), found %d", tftp_msg_type(buffer));
    return 1;
  }

  offset += 2;
  if (strlen(buffer+offset) > TFTP_MAX_FILENAME_LEN){
    LOG(LOG_ERR, "Filename too long (%d > %d): %s", (int) strlen(buffer+offset), TFTP_MAX_FILENAME_LEN, buffer+offset);
    return 3;
  }

  strcpy(filename, buffer+offset);
  offset += strlen(filename)+1;
  if (strlen(buffer+offset) > TFTP_MAX_MODE_LEN){
    LOG(LOG_ERR, "Mode string too long (%d > %d): %s", (int) strlen(buffer+offset), TFTP_MAX_MODE_LEN, buffer+offset);
    return 4;
  }

  strcpy(mode, buffer+offset);
  offset += strlen(mode)+1;
  if (buffer_len != offset){
    LOG(LOG_ERR, "Packet contains unexpected fields");
    return 2;
  }

  if (strcmp(mode, TFTP_STR_NETASCII) == 0 || strcmp(mode, TFTP_STR_OCTET) == 0)
    return 0;
  else{
    LOG(LOG_ERR, "Unrecognized transfer mode: %s", mode);
    return 5;
  }
}

int tftp_msg_get_size_wrq(char* filename, char* mode){
  return 4 + strlen(filename) + strlen(mode);
}

void tftp_msg_build_data(int block_n, char* data, int data_size, char* buffer){
  buffer[0] = 0;
  buffer[1] = 3;
  buffer[2] = block_n >> 8;
  buffer[3] = block_n;
  buffer += 4;
  memcpy(buffer, data, data_size);
}

int tftp_msg_unpack_data(char* buffer, int buffer_len, int* block_n, char* data, int* data_size){
  if (tftp_msg_type(buffer) != TFTP_TYPE_DATA){
    LOG(LOG_ERR, "Expected DATA message (3), found %d", tftp_msg_type(buffer));
    return 1;
  }

  if (buffer_len < 4){
    LOG(LOG_ERR, "Packet size too small for DATA: %d > 4", buffer_len);
    return 2;
  }

  *block_n = (((int)buffer[2]) << 8) + buffer[3];
  *data_size = buffer_len - 4;
  if (*data_size > 0)
    memcpy(data, buffer+4, *data_size);
  return 0;
}

int tftp_msg_get_size_data(int data_size){
  return data_size + 4;
}

void tftp_msg_build_ack(int block_n, char* buffer){
  buffer[0] = 0;
  buffer[1] = 4;
  buffer[2] = block_n >> 8;
  buffer[3] = block_n;
}

int tftp_msg_unpack_ack(char* buffer, int buffer_len, int* block_n){
  if (tftp_msg_type(buffer) != TFTP_TYPE_ACK){
    LOG(LOG_ERR, "Expected ACK message (4), found %d", tftp_msg_type(buffer));
    return 1;
  }

  if (buffer_len != 4){
    LOG(LOG_ERR, "Wrong packet size for ACK: %d != 4", buffer_len);
    return 2;
  }
  *block_n = (((int)buffer[2]) << 8) + buffer[3];
  return 0;
}

int tftp_msg_get_size_ack(){
  return 4;
}

void tftp_msg_build_error(int error_code, char* error_msg, char* buffer){
  buffer[0] = 0;
  buffer[1] = 5;
  buffer[2] = error_code >> 8;
  buffer[3] = error_code;
  buffer += 4;
  strcpy(buffer, error_msg);
}

int tftp_msg_unpack_error(char* buffer, int buffer_len, int* error_code, char*
error_msg){
  if (tftp_msg_type(buffer) != TFTP_TYPE_ERROR){
      LOG(LOG_ERR, "Expected ERROR message (5), found %d", tftp_msg_type(buffer));
      return 1;
    }

    *error_code = (((int)buffer[2]) << 8) + buffer[3];
    if (*error_code < 0 || *error_code > 7){
      LOG(LOG_ERR, "Unrecognized error code: %d", *error_code);
      return 4;
    }

    buffer += 4;
    if(strlen(buffer) > TFTP_MAX_ERROR_LEN){
      LOG(LOG_ERR, "Error string too long (%d > %d): %s", (int) strlen(buffer), TFTP_MAX_ERROR_LEN, buffer)
;
      return 3;
    }

    strcpy(error_msg, buffer);
    if (buffer_len != strlen(error_msg)+5){
      LOG(LOG_WARN, "Packet contains unexpected fields");
      return 2;
    }
    return 0;
}

int tftp_msg_get_size_error(char* error_msg){
  return 5 + strlen(error_msg);
}
```

## 4.9 tftp_msgs.h File Reference

Contructor for TFTP messages.

### Macros

- #define TFTP_TYPE_RRQ 1

  *Read request.*
- #define TFTP_TYPE_WRQ 2

  *Write request.*
- #define TFTP_TYPE_DATA 3

  *Data.*
- #define TFTP_TYPE_ACK 4

  *Acknowledgment.*
- #define TFTP_TYPE_ERROR 5

  *Error.*
- #define **TFTP_STR_NETASCII** "netascii"
- #define **TFTP_STR_OCTET** "octet"
- #define **TFTP_MAX_FILENAME_LEN** 255
- #define **TFTP_MAX_MODE_LEN** 8
- #define **TFTP_MAX_ERROR_LEN** 255
- #define **TFTP_DATA_BLOCK** 512

### Functions

- int tftp_msg_type (char ∗buffer)

    *Retuns msg type given message buffer.*
- void tftp_msg_build_rrq (char ∗filename, char ∗mode, char ∗buffer)

    *Builds a read request message.*
- int tftp_msg_unpack_rrq (char ∗buffer, int buffer_len, char ∗filename, char ∗mode)

    *Unpacks a read request message.*
- int tftp_msg_get_size_rrq (char ∗filename, char ∗mode)

    *Returns size in bytes of a read request message.*
- void tftp_msg_build_wrq (char ∗filename, char ∗mode, char ∗buffer)

    *Builds a write request message.*
- int tftp_msg_get_size_wrq (char ∗filename, char ∗mode)

    *Returns size in bytes of a write request message.*
- void tftp_msg_build_data (int block_n, char ∗data, int data_size, char ∗buffer)

    *Builds a data message.*
- int tftp_msg_unpack_data (char ∗buffer, int buffer_len, int ∗block_n, char ∗data, int ∗data_size)

    *Unpacks a data message.*
- int tftp_msg_get_size_data (int data_size)

    *Returns size in bytes of a data message.*
- void tftp_msg_build_ack (int block_n, char ∗buffer)

    *Builds an acknowledgment message.*
- int tftp_msg_unpack_ack (char ∗buffer, int buffer_len, int ∗block_n)

    *Unpacks an acknowledgment message.*
- int tftp_msg_get_size_ack ()

    *Returns size in bytes of an acknowledgment message.*
- void tftp_msg_build_error (int error_code, char ∗error_msg, char ∗buffer)

    *Builds an error message.*
- int tftp_msg_unpack_error (char ∗buffer, int buffer_len, int ∗error_code, char ∗error_msg)

    *Unpacks an error message.*
- int tftp_msg_get_size_error (char ∗error_msg)

    *Returns size in bytes of an error message.*

### 4.9.1 Detailed Description

Contructor for TFTP messages.

**Author**

Riccardo Mancini This library provides functions for building TFTP messages. There are 5 types of messages:

- 1: Read request (RRQ)
- 2: Write request (WRQ)
- 3: Data (DATA)
- 4: Acknowledgment (ACK)
- 5: Error (ERROR)

Definition in file tftp_msgs.h.

### 4.9.2 Function Documentation

#### 4.9.2.1 tftp_msg_build_ack()

```
void tftp_msg_build_ack (
             int block_n,
             char * buffer )
```

Builds an acknowledgment message.

Message format:

```
 2 bytes    2 bytes
 -------------------
| 04     |   Block #  |
 --------------------
```

**Parameters**

| | |
|---|---|
| *block↩_n* | block sequence number |
| *buffer* | data buffer where to build the message |

Definition at line 144 of file tftp_msgs.c.

#### 4.9.2.2 tftp_msg_build_data()

```
void tftp_msg_build_data (
             int block_n,
             char * data,
             int data_size,
             char * buffer )
```

Builds a data message.

Message format:

```
 2 bytes    2 bytes       n bytes
 ----------------------------------
| 03     |   Block #  |    Data     |
 ----------------------------------
```

**Parameters**

| | |
|---|---|
| *block_n* | block sequence number |
| *data* | pointer to the buffer containing the data to be transfered |
| *data_size* | data buffer size |
| *buffer* | data buffer where to build the message |

Definition at line 113 of file tftp_msgs.c.

#### 4.9.2.3 tftp_msg_build_error()

```
void tftp_msg_build_error (
             int error_code,
             char * error_msg,
             char * buffer )
```

Builds an error message.

Message format:

```
  2 bytes   2 bytes        string    1 byte
 ----------------------------------------
| 05     |  ErrorCode |    ErrMsg    |   0  |
 ----------------------------------------
```

**Parameters**

| | |
|---|---|
| *error_code* | error code (from 0 to 7) |
| *error_msg* | error message |
| *buffer* | data buffer where to build the message |

Definition at line 169 of file tftp_msgs.c.

#### 4.9.2.4 tftp_msg_build_rrq()

```
void tftp_msg_build_rrq (
             char * filename,
             char * mode,
             char * buffer )
```

Builds a read request message.

**2 bytes string 1 byte string 1 byte**

**| 01 | Filename | 0 | Mode | 0 |**

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |
| *buffer* | data buffer where to build the message |

Definition at line 18 of file tftp_msgs.c.

#### 4.9.2.5 tftp_msg_build_wrq()

```
void tftp_msg_build_wrq (
             char * filename,
             char * mode,
             char * buffer )
```

Builds a write request message.

Message format:

```
 2 bytes    string    1 byte    string    1 byte
 -----------------------------------------------
|   02   |  Filename  |   0  |    Mode     |   0  |
 -----------------------------------------------
```

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |
| *buffer* | data buffer where to build the message |

Definition at line 65 of file tftp_msgs.c.

#### 4.9.2.6 tftp_msg_get_size_ack()

```
int tftp_msg_get_size_ack ( )
```

Returns size in bytes of an acknowledgment message.

It just returns 4.

**Parameters**

| | |
|---|---|
| *data_size* | data buffer size |

**Returns**

size in bytes

Definition at line 165 of file tftp_msgs.c.

#### 4.9.2.7 tftp_msg_get_size_data()

```
int tftp_msg_get_size_data (
             int data_size )
```

Returns size in bytes of a data message.

It just sums 4 to data_size.

**Parameters**

| | |
|---|---|
| *data_size* | data buffer size |

**Returns**

size in bytes

Definition at line 140 of file tftp_msgs.c.

#### 4.9.2.8 tftp_msg_get_size_error()

```
int tftp_msg_get_size_error (
             char * error_msg )
```

Returns size in bytes of an error message.

**Parameters**

| | |
|---|---|
| *error_msg* | error message |

**Returns**

size in bytes

Definition at line 204 of file tftp_msgs.c.

#### 4.9.2.9 tftp_msg_get_size_rrq()

```
int tftp_msg_get_size_rrq (
             char * filename,
             char * mode )
```

Returns size in bytes of a read request message.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |

**Returns**

size in bytes

Definition at line 61 of file tftp_msgs.c.

#### 4.9.2.10 tftp_msg_get_size_wrq()

```
int tftp_msg_get_size_wrq (
             char * filename,
             char * mode )
```

Returns size in bytes of a write request message.

**Parameters**

| | |
|---|---|
| *filename* | name of the file |
| *mode* | requested transfer mode ("netascii" or "octet") |

**Returns**

size in bytes

Definition at line 109 of file tftp_msgs.c.

#### 4.9.2.11 tftp_msg_type()

```
int tftp_msg_type (
             char * buffer )
```

Retuns msg type given message buffer.

**Parameters**

| | |
|---|---|
| *buffer* | the buffer |

**Returns**

message type

**See also**

TFTP_TYPE_RRQ
TFTP_TYPE_WRQ
TFTP_TYPE_DATA
TFTP_TYPE_ACK
TFTP_TYPE_ERROR

Definition at line 14 of file tftp_msgs.c.

#### 4.9.2.12 tftp_msg_unpack_ack()

```
int tftp_msg_unpack_ack (
             char * buffer,
             int buffer_len,
             int * block_n )
```

Unpacks an acknowledgment message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *block_n* | pointer where block_n will be written [out] |
| *data* | pointer inside buffer where the data is [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 151 of file tftp_msgs.c.

#### 4.9.2.13 tftp_msg_unpack_data()

```
int tftp_msg_unpack_data (
             char * buffer,
             int buffer_len,
             int * block_n,
             char * data,
             int * data_size )
```

Unpacks a data message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *block_n* | pointer where block_n will be written [out] |
| *data* | pointer where to copy data [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 122 of file tftp_msgs.c.

#### 4.9.2.14 tftp_msg_unpack_error()

```
int tftp_msg_unpack_error (
             char * buffer,
             int buffer_len,
             int * error_code,
             char * error_msg )
```

Unpacks an error message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *error_code* | pointer where error_code will be written [out] |
| *error_msg* | pointer to error message inside the message [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 178 of file tftp_msgs.c.

#### 4.9.2.15 tftp_msg_unpack_rrq()

```
int tftp_msg_unpack_rrq (
            char * buffer,
            int buffer_len,
            char * filename,
            char * mode )
```

Unpacks a read request message.

Unpacks a write request message.

**Parameters**

| | |
|---|---|
| *buffer* | data buffer where the message to read is [in] |
| *buffer_len* | length of the buffer [in] |
| *filename* | name of the file [out] |
| *mode* | requested transfer mode ("netascii" or "octet") [out] |

**Returns**

0 if success, 1 if wrong opcode, 2 otherwise

Definition at line 27 of file tftp_msgs.c.

## 4.10 tftp_msgs.h

```

#ifndef TFTP_MSGS
#define TFTP_MSGS

#define TFTP_TYPE_RRQ   1

#define TFTP_TYPE_WRQ   2

#define TFTP_TYPE_DATA  3

#define TFTP_TYPE_ACK   4

```

```
#define TFTP_TYPE_ERROR 5

#define TFTP_STR_NETASCII "netascii"
#define TFTP_STR_OCTET "octet"

#define TFTP_MAX_FILENAME_LEN 255
#define TFTP_MAX_MODE_LEN 8
#define TFTP_MAX_ERROR_LEN 255

#define TFTP_DATA_BLOCK 512


int tftp_msg_type(char *buffer);


void tftp_msg_build_rrq(char* filename, char* mode, char* buffer);

int tftp_msg_unpack_rrq(char* buffer, int buffer_len, char* filename, char*
    mode);

int tftp_msg_get_size_rrq(char* filename, char* mode);

void tftp_msg_build_wrq(char* filename, char* mode, char* buffer);

int tftp_msg_unpack_rrq(char* buffer, int buffer_len, char* filename, char*
    mode);

int tftp_msg_get_size_wrq(char* filename, char* mode);

void tftp_msg_build_data(int block_n, char* data, int data_size, char* buffer);

int tftp_msg_unpack_data(char* buffer, int buffer_len, int* block_n, char* data, int*
    data_size);

int tftp_msg_get_size_data(int data_size);

void tftp_msg_build_ack(int block_n, char* buffer);

int tftp_msg_unpack_ack(char* buffer, int buffer_len, int* block_n);

int tftp_msg_get_size_ack();

void tftp_msg_build_error(int error_code, char* error_msg, char* buffer);

int tftp_msg_unpack_error(char* buffer, int buffer_len, int* error_code, char*
    error_msg);

int tftp_msg_get_size_error(char* error_msg);

#endif
```

## 4.11 tftp_server.c File Reference

Implementation of the TFTP server serving only read requests.

```
#include "include/tftp_msgs.h"
#include "include/tftp.h"
#include "include/fblock.h"
#include "include/inet_utils.h"
#include "include/debug_utils.h"
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "include/logging.h"
#include <unistd.h>
#include <time.h>
```

### Macros

- #define **LOG_LEVEL** LOG_INFO

### Functions

- void **print_help** ()
- int **send_file** (char ∗filename, char ∗mode, struct sockaddr_in ∗cl_addr)
- int **main** (int argc, char ∗∗argv)

### 4.11.1 Detailed Description

Implementation of the TFTP server serving only read requests.

**Author**

Riccardo Mancini

Definition in file tftp_server.c.

## 4.12 tftp_server.c

```
#define LOG_LEVEL LOG_INFO

#include "include/tftp_msgs.h"
#include "include/tftp.h"
#include "include/fblock.h"
#include "include/inet_utils.h"
#include "include/debug_utils.h"
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "include/logging.h"
#include <sys/types.h>
#include <unistd.h>
#include <time.h>

void print_help(){
  printf("Usage: ./tftp_server LISTEN_PORT FILES_DIR\n");
  printf("Example: ./tftp_server 69 .\n");
}


int send_file(char* filename, char* mode, struct sockaddr_in *cl_addr){
  struct sockaddr_in my_addr;
  int sd;
  int ret, tid;
  struct fblock m_fblock;

  sd = socket(AF_INET, SOCK_DGRAM, 0);
  my_addr = make_my_sockaddr_in(0);
  tid = bind_random_port(sd, &my_addr);
  if (tid == 0){
    LOG(LOG_ERR, "Could not bind to random port");
    perror("Could not bind to random port:");
    fblock_close(&m_fblock);
    return 2;
  } else
    LOG(LOG_INFO, "Bound to port %d", tid);

  if (strcmp(mode, TFTP_STR_OCTET) == 0){
    m_fblock = fblock_open(filename, TFTP_DATA_BLOCK, FBLOCK_READ|
FBLOCK_MODE_BINARY);
```

```
  } else if (strcmp(mode, TFTP_STR_NETASCII) == 0){
    m_fblock = fblock_open(filename, TFTP_DATA_BLOCK, FBLOCK_READ|
FBLOCK_MODE_TEXT);
  } else{
    LOG(LOG_ERR, "Unknown mode: %s", mode);
    return 2;
  }

  if (m_fblock.file == NULL){
    LOG(LOG_WARN, "Error opening file. Not found?");
    tftp_send_error(1, "File not found.", sd, cl_addr);
    return 1;
  }

  LOG(LOG_INFO, "Sending file...");

  ret = tftp_send_file(&m_fblock, sd, cl_addr);
  fblock_close(&m_fblock);

  if (ret != 0){
    LOG(LOG_ERR, "Error sending file: %d", ret);
    return 16+ret;
  }

  LOG(LOG_INFO, "File sent successfully");
  return 0;
}

int main(int argc, char** argv){
  short int my_port;
  char *directory, *filename, *path, *mode;
  int ret, max_msglen, type, len;
  char* in_buffer;
  unsigned int addrlen;
  int sd;
  struct sockaddr_in my_addr, cl_addr;
  int pid;

  if (argc != 3){
    print_help();
    return 1;
  }

  my_port = atoi(argv[1]);
  directory = argv[2];

  max_msglen = TFTP_MAX_MODE_LEN+TFTP_MAX_FILENAME_LEN+4;
  in_buffer = malloc(max_msglen);
  addrlen = sizeof(cl_addr);

  sd = socket(AF_INET, SOCK_DGRAM, 0);
  my_addr = make_my_sockaddr_in(my_port);
  ret = bind(sd, (struct sockaddr*) &my_addr, sizeof(my_addr));
  if (ret == -1){
    LOG(LOG_ERR, "Could not bind");
    return 1;
  }

  LOG(LOG_INFO, "Server is running");

  while (1){
    len = recvfrom(sd, in_buffer, max_msglen, 0, (struct sockaddr*)&cl_addr, &addrlen);
    type = tftp_msg_type(in_buffer);
    LOG(LOG_DEBUG, "Received message with type %d", type);
    if (type == TFTP_TYPE_RRQ){
      pid = fork();
      if (pid != 0){
        LOG(LOG_INFO, "Received RRQ, spawned new process %d", (int) pid);
        continue;
      }

      //init random seed
      srand(time(NULL));

      filename = malloc(TFTP_MAX_FILENAME_LEN);
      mode = malloc(TFTP_MAX_MODE_LEN);
      path = malloc(TFTP_MAX_FILENAME_LEN+strlen(directory));
      ret = tftp_msg_unpack_rrq(in_buffer, len, filename, mode);
      path[0] = '\0';
      strcat(path, directory);
      strcat(path, "/");
      strcat(path, filename);

      LOG(LOG_INFO, "User wants to read file %s in mode %s", filename, mode);

      ret = send_file(path, mode, &cl_addr);
      if (ret != 0)
```

```
        LOG(LOG_WARN, "Write terminated with an error: %d", ret);
      break;
    } else{
      LOG(LOG_WARN, "Wrong op code: %d", type);
      tftp_send_error(4, "Illegal TFTP operation.", sd, &cl_addr);
    }
  }

  LOG(LOG_INFO, "Exiting process %d", (int) getpid());
  return 0;
}
```

## Index