Four-in-a-Row

Generated by Doxygen 1.8.17

1 Four-in-a-row

Four-in-a-row game for the course of Foundations of Cyber Security at University of Pisa

2 TODO

Check for full board when we play without winning

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

5 File Index

5.1 File List

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

6 Data Structure Documentation

6.1 ClientSocketWrapper Class Reference

SocketWrapper for a TCP client.

```
#include <socket_wrapper.h>
```

Inherits SocketWrapper.

Public Member Functions

- int connectServer (Host host)
  Connects to a remote server.

Additional Inherited Members

6.1.1 Detailed Description

SocketWrapper for a TCP client.

It provides a new function to connect to server.

Definition at line 110 of file socket_wrapper.h.

6.1.2 Member Function Documentation

6.1.2.1 connectServer() `int ClientSocketWrapper::connectServer ( Host host )`

Connects to a remote server.

Returns

0 in case of success, something else otherwise

Definition at line 85 of file socket_wrapper.cpp.

6.2 Connect4 Class Reference

Public Member Functions

- Connect4 (int rows=6, int columns=7)

  Construct a new Connect 4 object.
- int getNumCols ()

  Get the number of columns of the board.
- int8_t play (int column, char player=0)

  Inserts a token.
- bool checkWin (int starting_row, int starting_col, char player=0)

  Checks if an inserted token causes a win.
- bool setPlayer (char player)

  Sets the default player.
- char getPlayer ()

  Get the default player.
- char getAdv ()

  Get the adversary, when a default player is set.
- void print (std::ostream &os)

  Prints the board.

Friends

- std::ostream & **operator**<< (std::ostream &os, const Connect4 &b)

6.2.1 Detailed Description

Definition at line 18 of file connect4.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Connect4()

```
Connect4::Connect4 (
        int rows = 6,
        int columns = 7 )
```

Construct a new Connect 4 object.

Parameters

rows	Number of rows
columns	Number of columns

Definition at line 19 of file connect4.cpp.

6.2.3 Member Function Documentation

6.2.3.1 checkWin()

```
bool Connect4::checkWin (
        int starting_row,
        int starting_col,
        char player = 0 )
```

Checks if an inserted token causes a win.

Parameters

starting_row	row of the token
starting_col	col of the token
player	marker of the player inserting the token

Returns

true if winning, false otherwise

Definition at line 83 of file connect4.cpp.

6.2.3.2 getAdv()

```
char Connect4::getAdv ( )
```

Get the adversary, when a default player is set.

Returns

enemy marker

Definition at line 146 of file connect4.cpp.

6.2.3.3 getNumCols()

```
int Connect4::getNumCols ( )
```

Get the number of columns of the board.

Returns

number of columns

Definition at line 128 of file connect4.cpp.

6.2.3.4 getPlayer() `char Connect4::getPlayer ( )`

Get the default player.

Returns

player marker

Definition at line 142 of file connect4.cpp.

6.2.3.5 play()

```
int8_t Connect4::play (
        int column,
        char player = 0 )
```

Inserts a token.

Parameters

column	target column where the token should be added
player	player who is making the move

Return values

1	Success with win
0	Success without win
-1	Failure for full column
-2	Board is full, it could be so before or after the move takes place

Definition at line 31 of file connect4.cpp.

6.2.3.6 print()

```
void Connect4::print (
        std::ostream & os )
```

Prints the board.

Parameters

os	Output stream where the board has to be printed

Definition at line 15 of file connect4.cpp.

6.2.3.7 setPlayer()

```
bool Connect4::setPlayer (
        char player )
```

Sets the default player.

Parameters

player	player to be set

Returns

true if a valid player was supplied and set

false otherwise

Definition at line 132 of file connect4.cpp.

6.3 Host Class Reference

Class that holds a host information.

```
#include <host.h>
```

Public Member Functions

- Host (struct sockaddr_in addr)

  Constructs new instance from given inet address.
- Host (char ∗ip, int port)

  Constructs new instance from IP/port pair.
- struct sockaddr_in getAddress ()

  Returns the inet address of the host.
- string toString ()

  Returns the inet address of the host.

6.3.1 Detailed Description

Class that holds a host information.

At the moment, it only holds its inet addr but in the future its public key and other OpenSSL stuff will be put here.

Definition at line 23 of file host.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Host() [1/2]

```
Host::Host (
            struct sockaddr_in addr )  [inline]
```

Constructs new instance from given inet address.

Parameters

addr	the inet address of the remote host

Definition at line 33 of file host.h.

6.3.2.2 Host() [2/2]

```
Host::Host (
        char * ip,
        int port )  [inline]
```

Constructs new instance from IP/port pair.

Parameters

addr	the inet address of the remote host

Definition at line 41 of file host.h.

6.4 Message Class Reference

Abstract class for Messages.

```
#include <messages.h>
```

Inherited by MoveMessage, and StartGameMessage.

Public Member Functions

- virtual int write (char ∗buffer)=0

  Write message to buffer.
- virtual int read (char ∗buffer, int len)=0

  Read message from buffer.
- virtual size_t size ()=0

  Get required buffer size.
- virtual string getName ()=0

  Get message name (for debug purposes)
- virtual MessageType **getType** ()=0

6.4.1 Detailed Description

Abstract class for Messages.

Definition at line 37 of file messages.h.

6.4.2 Member Function Documentation

6.4.2.1 read()

```
virtual int Message::read (
        char * buffer,
        int len )  [pure virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implemented in MoveMessage, and StartGameMessage.

6.4.2.2 write()

```
virtual int Message::write (
        char * buffer )  [pure virtual]
```

Write message to buffer.

Returns

number of bytes written

Implemented in MoveMessage, and StartGameMessage.

6.5 MoveMessage Class Reference

Message that signals a move.

```
#include <messages.h>
```

Inherits Message.

Public Member Functions

- **MoveMessage** (char col)
- int write (char ∗buffer)

  Write message to buffer.
- int read (char ∗buffer, int len)

  Read message from buffer.
- size_t size ()

  Get required buffer size.
- string getName ()

  Get message name (for debug purposes)
- char **getColumn** ()
- MessageType **getType** ()

6.5.1 Detailed Description

Message that signals a move.

Definition at line 87 of file messages.h.

6.5.2 Member Function Documentation

6.5.2.1 read()

```
int MoveMessage::read (
    char * buffer,
    int len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 60 of file messages.cpp.

6.5.2.2 write()

```
int MoveMessage::write (
    char * buffer )  [virtual]
```

Write message to buffer.

Returns

number of bytes written

Implements Message.

Definition at line 54 of file messages.cpp.

6.6 ServerSocketWrapper Class Reference

SocketWrapper for a TCP server.

```
#include <socket_wrapper.h>
```

Inherits SocketWrapper.

Public Member Functions

- ServerSocketWrapper ()

  Initialize a new socket on a random port.
- ServerSocketWrapper (int port)

  Initialize a new socket at the requested port.
- SocketWrapper ∗ acceptClient ()

  Accepts any incoming connection and returns the related SocketWrapper.
- int getPort ()

  Returns port the server is listening new connections on.

Additional Inherited Members

6.6.1 Detailed Description

SocketWrapper for a TCP server.

It provides a new function to accept clients. Constructor also set listen mode.

Definition at line 126 of file socket_wrapper.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 ServerSocketWrapper() `[1/2]` `ServerSocketWrapper::ServerSocketWrapper ( )`

Initialize a new socket on a random port.

Parameters

port	the port you want to bind on

Definition at line 106 of file socket_wrapper.cpp.

6.6.2.2 ServerSocketWrapper() `[2/2]` `ServerSocketWrapper::ServerSocketWrapper ( int port )`

Initialize a new socket at the requested port.

Parameters

port	the port you want to bind on

Definition at line 121 of file socket_wrapper.cpp.

6.7 SocketWrapper Class Reference

Wrapper class around sockaddr_in and socket descriptor.

`#include <socket_wrapper.h>`

Inherited by ClientSocketWrapper, and ServerSocketWrapper.

Public Member Functions

- SocketWrapper ()
  Initialize on a new socket.
- SocketWrapper (int sd)
  Initialize using existing socket.
- int getDescriptor ()
  Returns current socket file descriptor.
- Message ∗ receiveAnyMsg (size_t size=MAX_MSG_SIZE)
  Receive any new message from the socket.
- Message ∗ receiveMsg (MessageType type, size_t size=MAX_MSG_SIZE)
  Receive a new message of the given type from the socket.
- Message ∗ receiveMsg (MessageType type[ ], int n_types, size_t size=MAX_MSG_SIZE)
  Receive a new message of any of the given types from the socket.
- int sendMsg (Message ∗msg)
  Sends the given message to the peer host through the socket.
- void setOtherAddr (struct sockaddr_in addr)
  Sets the address of the other host.
- Host getConnectedHost ()
  Returns connected host.

Protected Attributes

- struct sockaddr_in other_addr
  Other host inet socket.
- int socket_fd
  Socket file descriptor.

6.7.1 Detailed Description

Wrapper class around sockaddr_in and socket descriptor.

It provides a more simple interface saving a lot of boiler-plate code. There are two subclasses: ClientSocketWrapper and ServerSocketWrapper.

Definition at line 24 of file socket_wrapper.h.

6.7.2 Member Function Documentation

6.7.2.1 receiveAnyMsg() `Message ∗ SocketWrapper::receiveAnyMsg ( size_t size = MAX_MSG_SIZE )`

Receive any new message from the socket.

This API is blocking.

Parameters

size	the size of the temporary buffer

Returns

the received message or null if an error occurred

Definition at line 22 of file socket_wrapper.cpp.

6.7.2.2 receiveMsg() [1/2]

```
Message * SocketWrapper::receiveMsg (
        MessageType type,
        size_t size = MAX_MSG_SIZE )
```

Receive a new message of the given type from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the type to keep
size	the size of the temporary buffer

Returns

the received message or null if an error occurred

Definition at line 37 of file socket_wrapper.cpp.

6.7.2.3 receiveMsg() [2/2]

```
Message * SocketWrapper::receiveMsg (
        MessageType type[ ],
        int n_types,
        size_t size = MAX_MSG_SIZE )
```

Receive a new message of any of the given types from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the types to keep (array)
n_types	the number of types to keep (array length)

Returns

the received message or null if an error occurred

Definition at line 41 of file socket_wrapper.cpp.

6.7.2.4 sendMsg()

```
int SocketWrapper::sendMsg (
        Message * msg )
```

Sends the given message to the peer host through the socket.

Parameters

msg	the message to be sent

Returns

0 in case of success, something else otherwise

Definition at line 60 of file socket_wrapper.cpp.

6.7.2.5 setOtherAddr()

```
void SocketWrapper::setOtherAddr (
        struct sockaddr_in addr )  [inline]
```

Sets the address of the other host.

This is used when initializing a new SocketWrapper for a newly accepter connection.

Definition at line 97 of file socket_wrapper.h.

6.8 StartGameMessage Class Reference

Message that signals to start a new game.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- int write (char ∗buffer)

  Write message to buffer.
- int read (char ∗buffer, int len)

  Read message from buffer.
- size_t size ()

  Get required buffer size.
- string getName ()

  Get message name (for debug purposes)
- MessageType **getType** ()

6.8.1 Detailed Description

Message that signals to start a new game.

Definition at line 70 of file messages.h.

6.8.2 Member Function Documentation

6.8.2.1 read()

```
int StartGameMessage::read (
        char * buffer,
        int len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 50 of file messages.cpp.

6.8.2.2 write()

```
int StartGameMessage::write (
        char * buffer )  [virtual]
```

Write message to buffer.

Returns

number of bytes written

Implements Message.

Definition at line 45 of file messages.cpp.

7 File Documentation

7.1 client.cpp File Reference

Implementation of a 4-in-a-row game.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"
#include "network/socket_wrapper.h"
#include "network/host.h"
```

Macros

- #define **MY_TURN** (0)
- #define **THEIR_TURN** (1)

Functions

- void print_help ()

  Prints command usage information.
- void **printWelcome** ()
- int **main** (int argc, char ∗∗argv)

Variables

- char **players** [ ] = {'X', 'O'}

7.1.1 Detailed Description

Implementation of a 4-in-a-row game.

Author

Mirko Laruina

Date

2020-05-14

Definition in file client.cpp.

7.2 client.cpp

```

#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"
#include "network/socket_wrapper.h"
#include "network/host.h"

using namespace std;

#define MY_TURN    (0)
#define THEIR_TURN (1)

char players[] = {'X', 'O'};

void print_help(){
  cout«"On host A: ./client"«endl;
  cout«"On host B: ./client ipA portA"«endl;
}

void printWelcome(){
    cout«"*****************************************************************\n"
        «"*        __ __     _                                           *\n"
        «"*       / // /    (_)____    ____ _   _________ _      __      *\n"
        «"*      / // /_   / / __ \\   / __ `/  / ___/ __ \\ | /| / /      *\n"
        «"*     /__  __/  / / / / /  / /_/ /  / /  / /_/ / |/ |/ /       *\n"
        «"*       /_/    /_/_/ /_/   \\__,_/  /_/   \\____/|__/|__/        *\n"
        «"*                                                              *\n"
        «"*****************************************************************"
        «endl;
}

int main(int argc, char** argv){
    // handle server selection on params here
    char in_buffer[256];
    int choosen_col;
    int adv_col;
    int win;
    int turn;
    SocketWrapper *sw;
    Host* peer;
    srand(time(NULL));

    printWelcome();
    cout«endl«"Welcome to 4-in-a-row!"«endl;
    cout«"The rules of the game are simple: you win when you have 4 connected tokens along any
 direction."«endl;

    Connect4 c;
    cout«"Who do you want to be? X or O ?"«endl;

    do {
        cout«"> ";
        cin.getline(in_buffer, sizeof(in_buffer));
    } while (!c.setPlayer(in_buffer[0]));
    cout«"You are playing as "«c.getPlayer()«endl;


    if (argc != 1 && argc != 2 && argc != 3){
        print_help();
        return 1;
    } else if (argc == 1 || argc == 2){ // wait for peer connection
        ServerSocketWrapper *ssw;
        if (argc == 1){
            ssw = new ServerSocketWrapper();
        } else{
            ssw = new ServerSocketWrapper(atoi(argv[1]));
        }
        cout«"Waiting for connection on port: "«ssw->getPort()«endl;

        sw = ssw->acceptClient();

        Host p = sw->getConnectedHost();
        peer = &p;

        cout«"Accepted client: "«peer->toString()«endl;

        StartGameMessage *sgm = dynamic_cast<StartGameMessage*>(sw->receiveMsg(START_GAME));
        if (sgm == NULL){
            LOG(LOG_ERR, "Connection error");
            return 1;
        }

        LOG(LOG_INFO, "Connected to %s", peer->toString().c_str());
        turn = MY_TURN;
    } else if (argc == 3){
        peer = new Host(argv[1], atoi(argv[2]));

        cout«"Connecting to: "«peer->toString()«endl;

        ClientSocketWrapper *csw = new ClientSocketWrapper();

        int ret = csw->connectServer(*peer);

        if (ret != 0){
            LOG(LOG_ERR, "Connection error");
            return 1;
        }

        sw = csw;

        StartGameMessage m;
        ret = sw->sendMsg(&m);

        if (ret != 0){
            LOG(LOG_ERR, "Connection error");
            return 1;
        }

        LOG(LOG_INFO, "Connected to %s", peer->toString().c_str());
        turn = THEIR_TURN;
    }

    cout«"This is the starting board:"«endl;
    cout«c;
    do {
        if (turn == MY_TURN){
            cout«"Write the column you want to insert the token to"«endl;
            do {
                cout«"> ";
                cin.getline(in_buffer, sizeof(in_buffer));
                choosen_col = in_buffer[0]-'0';
            } while(choosen_col < 0 || choosen_col > 7);

            win = c.play(choosen_col-1, c.getPlayer());
            cout«c;

            if (win != -1){
                MoveMessage mm(choosen_col-1);
                int ret = sw->sendMsg(&mm);
                if (ret != 0){
                    LOG(LOG_ERR, "Connection error");
                    return 1;
                }
            }

            if(win == 1){
                cout«"Congratulation, you won!"«endl;
            } else if(win == -1){
                cout«"The column is full, choose a different one!"«endl;
                continue;
            } else if(win == -2){
                cout«"The entire board is filled: it is a draw!"«endl;
                break;
            } else{
                turn = THEIR_TURN;
            }

        } else{      // THEIR_TURN
            do {
                MoveMessage *mm;
                mm = dynamic_cast<MoveMessage*>(sw->receiveMsg(MOVE));
                if (mm == NULL){
                    LOG(LOG_ERR, "Connection error");
                    return 1;
                }
                adv_col = mm->getColumn();
                cout«"Your enemy has chosen column "«adv_col«endl;
                win = c.play(adv_col, c.getAdv());
                cout«c;
                if(win == 1){
                    cout«"Damn! You lost!"«endl;
                } else if(win == -1){
                    cout«"The column is full, the adversary has lost!"«endl;
                    break;
                } else if(win == -2){
                    cout«"The entire board is filled: it is a draw!"«endl;
                    break;
                }
            } while (win == -1);
            turn = MY_TURN;
        }
    } while (win == -1 || win == 0);
    return 0;
}
```

7.3 connect4.cpp File Reference

Implementation of connect4.h.

```
#include "connect4.h"
```

Functions

- ostream & **operator**<< (ostream &os, const Connect4 &c)

7.3.1 Detailed Description

Implementation of connect4.h.

Author

Mirko Laruina

Date

2020-05-14

See also

connect4.h

Definition in file connect4.cpp.

7.4 connect4.cpp

```

#include "connect4.h"
using namespace std;

void Connect4::print(ostream& os){
    os<<*this;
}

Connect4::Connect4(int rows /* = 6 */, int columns /* = 7 */){
    rows_ = rows;
    cols_ = columns;
    size_ = rows*columns;
    full_ = false;

    //Maybe check for overflow if we will use different board values

    cells_ = new char[size_];
    memset(cells_, 0, size_);
}

int8_t Connect4::play(int col, char player){
    // bool col_full = true;
    if(player == 0){
        player = player_;
    }

    //Trying to play with a full board
    if(full_){
        return -2;
    }

    for(int i = rows_-1; i>=0; --i){
        if(cells_[i*cols_+col] == 0){
            // col_full = false;
            cells_[i*cols_+col] = player;
            if( checkWin(i, col, player) ){
                return 1;
            } else {
                //All the board could be full now
                if(i == 0 && checkFullTopRow()){
                    full_ = true;
                    return -2;
                } else {
                    return 0;
                }
            }
        }
    }

    //We are sure the board is not full, otherwise we would have already exited
    //If a play was possible, we would have already exited too
    //Only possible case is full column
    return -1;
}

int Connect4::countNexts(char player, int row, int col, int di, int dj){
    int count = 0;
    for(
        int i = row+di, j = col+dj;
        i >= 0 && j >= 0 && i < rows_ && j < cols_;
        i+=di, j+=dj)
    {
        if(cells_[i*cols_+j] != player){
            break;
        } else {
            LOG(LOG_DEBUG, "%d %d", i, j);
            count++;
        }
    }
    return count;
}

bool Connect4::checkWin(int row, int col, char player){
    /*
        Take any of the 4 possible directions
        count how many token of the same player there are
        before and after the new one
        if more than 4, declare win
    */

    LOG(LOG_DEBUG, "Checking (%d, %d)", row, col);
    if(player == 0){
        player = player_;
    }

    for(int di = 1; di >= 0 && di != -1; --di){
        for(int dj = 1; dj >= 0 && di != -1; --dj){
            // direction (0, 0) is useless, since we would miss diagonal (-1, 1)
            // we can exploit the loop to iterate over that
            if(di == 0 && dj == 0){
                di = -1;
                dj = 1;
            }

            int count_forward = Connect4::countNexts(player, row, col, di, dj);
            int count_backward = Connect4::countNexts(player, row, col, -di, -dj);

            // N_IN_A_ROW minus 1 since the token just inserted is excluded
            if(count_forward + count_backward >= (N_IN_A_ROW - 1)){
                return true;
            }
        }
    }

    return false;

}

bool Connect4::checkFullTopRow(){
    for(int j = 0; j<cols_; ++j){
        if(cells_[j] == 0){
            return false;
        }
    }
    return true;
}

int Connect4::getNumCols(){
    return cols_;
}

bool Connect4::setPlayer(char player){
    if(player == 'X' || player == 'x'
        || player == 'O' || player == 'o'){
        player_ = toupper(player);
        adversary_ = player_ == 'X' ? 'O' : 'X';
        return true;
    }
    return false;
}

char Connect4::getPlayer(){
    return player_;
}

char Connect4::getAdv(){
    return adversary_;
}

```

```
ostream& operator«(ostream& os, const Connect4& c){
    int width = 2+3*(c.rows_+1);
    for(int i = 0; i<width; ++i){
        os«'*';
    }
    os«endl;

    for(int i = 0; i<c.rows_; ++i){
        os«"*";
        for(int j = 0; j<c.cols_; ++j){
            if(c.cells_[i*c.cols_+j] == 0){
                os«"   ";
            } else {
                os« " " « (c.cells_[i*c.cols_+j] == 'X' ? "\033[31mX" : "\033[34mO") «" ";
            }
        }
        os«"\033[0m*"«endl;
    }

    for(int i = 0; i<width; ++i){
        os«'*';
    }
    os«endl;

    for(int i = 0; i<width; ++i){
        if( (i+1)%3 == 0  ){
            os«(i+1)/3;
        } else {
            os«" ";
        }
    }
    os«endl;
    return os;
}
```

7.5 connect4.h File Reference

Header file for the class responsible of handling the board of a Connect4 game.

```
#include <iostream>
#include <cstring>
#include "logging.h"
```

Data Structures

- class Connect4

Macros

- #define **N_IN_A_ROW** 4

7.5.1 Detailed Description

Header file for the class responsible of handling the board of a Connect4 game.

Author

Mirko Laruina

Date

2020-05-14

Definition in file connect4.h.

7.6 connect4.h

```
#ifndef CONNECT4_H
#define CONNECT4_H
#include <iostream>
#include <cstring>
#include "logging.h"

#define N_IN_A_ROW 4

class Connect4 {
    int rows_, cols_, size_;

    bool full_;

    char* cells_;

    char player_;

    char adversary_;

    int countNexts(char player, int row, int col, int di, int dj);

    bool checkFullTopRow();
    public:

    Connect4(int rows = 6, int columns = 7);

    int getNumCols();

    int8_t play(int column, char player = 0);

    bool checkWin(int starting_row, int starting_col, char player = 0);

    bool setPlayer(char player);

    char getPlayer();

    char getAdv();

    void print(std::ostream& os);

    friend std::ostream& operator«(std::ostream& os, const Connect4& b);
};
#endif //CONNECT4_H
```

7.7 dump_buffer.cpp File Reference

Implementation of dump_buffer.h.

```
#include "utils/dump_buffer.h"
#include "logging.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- void dump_buffer_hex (char ∗buffer, int len)

  Prints content of buffer to stdout, showing it as hex values.

7.7.1 Detailed Description

Implementation of dump_buffer.h.

Author

Riccardo Mancini

See also

dump_buffer.h

Definition in file dump_buffer.cpp.

7.7.2 Function Documentation

7.7.2.1 dump_buffer_hex() `void dump_buffer_hex ( char * buffer, int len )`

```
void dump_buffer_hex (
            char * buffer,
            int len )
```

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

Parameters

buffer	pointer to the buffer to be printed
len	the length (in bytes) of the buffer

Definition at line 17 of file dump_buffer.cpp.

7.8 dump_buffer.cpp

```
00001
00010 #include "utils/dump_buffer.h"
00011 #include "logging.h"
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015
00016
00017 void dump_buffer_hex(char* buffer, int len){
00018   char *str, tmp[4];
00019   int i;
00020
00021   str = (char*) malloc(len*3+1);
00022
00023   str[0] = '\0';
00024   for (i=0; i<len; i++){
00025     sprintf(tmp, "%02x ", (unsigned char) buffer[i]);
00026     strcat(str, tmp);
00027   }
00028
00029   LOG(LOG_DEBUG, "%s", str);
00030   free(str);
00031 }
```

7.9 dump_buffer.h File Reference

Utility function for dumping a buffer as hex string.

Functions

- void dump_buffer_hex (char ∗buffer, int len)

  Prints content of buffer to stdout, showing it as hex values.

7.9.1 Detailed Description

Utility function for dumping a buffer as hex string.

Author

Riccardo Mancini

Date

2020-05-17

Definition in file dump_buffer.h.

7.9.2 Function Documentation

7.9.2.1 dump_buffer_hex()

```
void dump_buffer_hex (
        char * buffer,
        int len )
```

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

Parameters

buffer	pointer to the buffer to be printed
len	the length (in bytes) of the buffer

Definition at line 17 of file dump_buffer.cpp.

7.10 dump_buffer.h

```
00001
00010 #ifndef DUMP_BUFFER_H
00011 #define DUMP_BUFFER_H
00012
00013
00022 void dump_buffer_hex(char* buffer, int len);
00023
00024
00025 #endif // DUMP_BUFFER_H
```

7.11 host.h File Reference

Definition of the helper class "Host".

```
#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"
```

Data Structures

- class Host

  Class that holds a host information.

7.11.1 Detailed Description

Definition of the helper class "Host".

Author

Riccardo Mancini

Date

2020-05-17

Definition in file host.h.

7.12 host.h

```

#ifndef HOST_H
#define HOST_H

#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"

class Host{
private:
    struct sockaddr_in addr;

public:
    Host(struct sockaddr_in addr)
        : addr(addr) {}

    Host(char* ip, int port){
        addr = make_sv_sockaddr_in(ip, port);
    }

    struct sockaddr_in getAddress(){return addr;}

    string toString(){
        return sockaddr_in_to_string(addr);
    }

};

#endif // HOST_H
```

7.13 inet_utils.cpp File Reference

Implementation of inet_utils.h.

```
#include <stdlib.h>
#include <string>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "network/inet_utils.h"
#include "logging.h"
```

Functions

- int bind_random_port (int socket, struct sockaddr_in ∗addr)

    Binds socket to a random port.
- struct sockaddr_in make_sv_sockaddr_in (char ∗ip, int port)

    Makes sockaddr_in structure given ip string and port of server.
- struct sockaddr_in make_my_sockaddr_in (int port)

    Makes sockaddr_in structure of this host.
- int sockaddr_in_cmp (struct sockaddr_in sai1, struct sockaddr_in sai2)

    Compares INET addresses, returning 0 in case they're equal.
- string sockaddr_in_to_string (struct sockaddr_in src)

    Converts sockaddr_in structure to string to be printed.

7.13.1 Detailed Description

Implementation of inet_utils.h.

Author

Riccardo Mancini

See also

inet_utils.h

Date

2020-05-17

Definition in file inet_utils.cpp.

7.13.2 Function Documentation

7.13.2.1 bind_random_port()

```
int bind_random_port (
        int socket,
        struct sockaddr_in * addr )
```

Binds socket to a random port.

Parameters

socket	socket ID
addr	inet addr structure

Returns

0 in case of failure, port it could bind to otherwise

See also

FROM_PORT

TO_PORT

MAX_TRIES

Definition at line 24 of file inet_utils.cpp.

7.13.2.2 make_my_sockaddr_in() `struct sockaddr_in make_my_sockaddr_in ( int port )`

Makes sockaddr_in structure of this host.

INADDR_ANY is used as IP address.

Parameters

port	port of the server

Returns

sockaddr_in structure this host on given port

Definition at line 53 of file inet_utils.cpp.

7.13.2.3 make_sv_sockaddr_in() `struct sockaddr_in make_sv_sockaddr_in ( char * ip, int port )`

Makes sockaddr_in structure given ip string and port of server.

Parameters

ip	ip address of server
port	port of the server

Returns

sockaddr_in structure for the given server

Definition at line 44 of file inet_utils.cpp.

7.13.2.4 sockaddr_in_cmp()

```
int sockaddr_in_cmp (
        struct sockaddr_in sai1,
        struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

Parameters

sai1	first address
sai2	second address

Returns

0 if thery're equal, 1 otherwise

Definition at line 62 of file inet_utils.cpp.

7.13.2.5 sockaddr_in_to_string()

```
string sockaddr_in_to_string (
        struct sockaddr_in src )
```

Converts sockaddr_in structure to string to be printed.

Parameters

src	the input address
dst	the output string (must be at least MAX_SOCKADDR_STR_LEN long)

Definition at line 70 of file inet_utils.cpp.

7.14 inet_utils.cpp

```
00001
00012 #include <stdlib.h>
00013 #include <string>
00014 #include <string.h>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
00017 #include <arpa/inet.h>
00018
00019 #include "network/inet_utils.h"
00020 #include "logging.h"
00021
00022 using namespace std;
00023
00024 int bind_random_port(int socket, struct sockaddr_in *addr){
    int port, ret, i;
    for (i = 0; i < MAX_TRIES; i++){
        if (i == 0) // first I generate a random one
            port = rand() % (TO_PORT - FROM_PORT + 1) + FROM_PORT;
        else //if it's not free I scan the next one
            port = (port - FROM_PORT + 1) % (TO_PORT - FROM_PORT + 1) + FROM_PORT;

        LOG(LOG_DEBUG, "Trying port %d...", port);

        addr->sin_port = htons(port);
        ret = bind(socket, (struct sockaddr *)addr, sizeof(*addr));
        if (ret != -1)
            return port;
        // consider only some errors?
    }
    LOG(LOG_ERR, "Could not bind to random port after %d attempts", MAX_TRIES);
    return 0;
}

struct sockaddr_in make_sv_sockaddr_in(char *ip, int port){
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    inet_pton(AF_INET, ip, &addr.sin_addr);
    return addr;
}

struct sockaddr_in make_my_sockaddr_in(int port){
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    return addr;
}

int sockaddr_in_cmp(struct sockaddr_in sai1, struct sockaddr_in sai2){
    if (sai1.sin_port == sai2.sin_port &&
        sai1.sin_addr.s_addr == sai2.sin_addr.s_addr)
        return 0;
    else
        return 1;
}

string sockaddr_in_to_string(struct sockaddr_in src){
    char dst[MAX_SOCKADDR_STR_LEN];
    char port_str[6];
    const char *ret;

    sprintf(port_str, "%d", ntohs(src.sin_port));

    ret = inet_ntop(AF_INET, (void *)&src.sin_addr, dst, MAX_SOCKADDR_STR_LEN);
    if (ret != NULL){
        strcat(dst, ":");
        strcat(dst, port_str);
    } else {
        strcpy(dst, "ERROR");
    }

    string s = dst;

    return s;
}
```

7.15 inet_utils.h File Reference

Utility funcions for managing inet addresses.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string>
```

Macros

- #define FROM_PORT 49152

Random port will be greater or equal to FROM_PORT.

- #define TO_PORT 65535

  Random port will be lower or equal to TO_PORT.
- #define MAX_TRIES 256

  Maximum number of trials before giving up opening a random port.
- #define MAX_SOCKADDR_STR_LEN 22

  Maximum number of characters of INET address to string (eg 123.156.189.123:45678).

Functions

- int bind_random_port (int socket, struct sockaddr_in ∗addr)

  Binds socket to a random port.
- struct sockaddr_in make_sv_sockaddr_in (char ∗ip, int port)

  Makes sockaddr_in structure given ip string and port of server.
- struct sockaddr_in make_my_sockaddr_in (int port)

  Makes sockaddr_in structure of this host.
- int sockaddr_in_cmp (struct sockaddr_in sai1, struct sockaddr_in sai2)

  Compares INET addresses, returning 0 in case they're equal.
- string sockaddr_in_to_string (struct sockaddr_in src)

  Converts sockaddr_in structure to string to be printed.

7.15.1 Detailed Description

Utility funcions for managing inet addresses.

Author

Riccardo Mancini

This library provides functions for creating sockaddr_in structures from IP address string and integer port number and for binding to a random port (chosen using rand() builtin C function).

Date

2020-05-17

See also

sockaddr_in

rand

Definition in file inet_utils.h.

7.15.2 Function Documentation

7.15.2.1 bind_random_port()

```
int bind_random_port (
        int socket,
        struct sockaddr_in * addr )
```

Binds socket to a random port.

Parameters

socket	socket ID
addr	inet addr structure

Returns

0 in case of failure, port it could bind to otherwise

See also

FROM_PORT
TO_PORT
MAX_TRIES

Definition at line 24 of file inet_utils.cpp.

7.15.2.2 make_my_sockaddr_in()

```
struct sockaddr_in make_my_sockaddr_in (
        int port )
```

Makes sockaddr_in structure of this host.

INADDR_ANY is used as IP address.

Parameters

port	port of the server

Returns

sockaddr_in structure this host on given port

Definition at line 53 of file inet_utils.cpp.

7.15.2.3 make_sv_sockaddr_in()

```
struct sockaddr_in make_sv_sockaddr_in (
        char * ip,
        int port )
```

Makes sockaddr_in structure given ip string and port of server.

Parameters

ip	ip address of server
port	port of the server

Returns

sockaddr_in structure for the given server

Definition at line 44 of file inet_utils.cpp.

7.15.2.4 sockaddr_in_cmp()

```
int sockaddr_in_cmp (
        struct sockaddr_in sai1,
        struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

Parameters

sai1	first address
sai2	second address

Returns

0 if thery're equal, 1 otherwise

Definition at line 62 of file inet_utils.cpp.

7.15.2.5 sockaddr_in_to_string()

```
string sockaddr_in_to_string (
        struct sockaddr_in src )
```

Converts sockaddr_in structure to string to be printed.

Parameters

src	the input address
dst	the output string (must be at least MAX_SOCKADDR_STR_LEN long)

Definition at line 70 of file inet_utils.cpp.

7.16 inet_utils.h

```

#ifndef INET_UTILS
#define INET_UTILS


#include <sys/socket.h>
#include <netinet/in.h>
#include <string>

using namespace std;

#define FROM_PORT 49152

#define TO_PORT   65535
```

```
#define MAX_TRIES 256

#define MAX_SOCKADDR_STR_LEN 22


int bind_random_port(int socket, struct sockaddr_in *addr);

struct sockaddr_in make_sv_sockaddr_in(char* ip, int port);

struct sockaddr_in make_my_sockaddr_in(int port);

int sockaddr_in_cmp(struct sockaddr_in sai1, struct sockaddr_in sai2);

string sockaddr_in_to_string(struct sockaddr_in src);


#endif
```

7.17 logging.h File Reference

Logging macro.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

Macros

- #define **LOG_FATAL** (1)
- #define **LOG_ERR** (2)
- #define **LOG_WARN** (3)
- #define **LOG_INFO** (4)
- #define **LOG_DEBUG** (5)
- #define **LOG_LEVEL** LOG_DEBUG
- #define **LOG**(level, ...)

7.17.1 Detailed Description

Logging macro.

Author

Riccardo Mancini

This file contains a macro for logging in different levels.

There are 5 levels of logging:

- fatal (LOG_FATAL)
- error (LOG_ERROR)
- warning (LOG_WARN)
- information (LOG_INFO)
- debug (LOG_DEBUG)

The first three will be outputted to stderr, the latter two to stdout.

You can define a LOG_LEVEL for hiding some of the logging messages in a per-executable basis. In order to do so, you need to put

```
const int LOG_LEVEL = LOG_INFO;
```

in the file containing the main and

```
extern const int LOG_LEVEL;
```

in any other file using this macro.

Adapted from https://stackoverflow.com/a/328660

Definition in file logging.h.

7.18 logging.h

```

#ifndef LOGGING
#define LOGGING


#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>


#define LOG_FATAL    (1)
#define LOG_ERR      (2)
#define LOG_WARN     (3)
#define LOG_INFO     (4)
#define LOG_DEBUG    (5)

#ifndef LOG_LEVEL
#define LOG_LEVEL LOG_DEBUG
#endif


#define LOG(level, ...) do {  \
                          if (level <= LOG_LEVEL) { \
                            FILE *dbgstream; \
                            char where[35]; \
                            switch(level){ \
                              case LOG_FATAL: \
                                dbgstream = stderr; \
                                fprintf(dbgstream, "[FATAL]"); \
                                break; \
                              case LOG_ERR: \
                                dbgstream = stderr; \
                                fprintf(dbgstream, "[ERROR]"); \
                                break; \
                              case LOG_WARN: \
                                dbgstream = stderr; \
                                fprintf(dbgstream, "[WARN ]"); \
                                break; \
                              case LOG_INFO: \
                                dbgstream = stdout; \
                                fprintf(dbgstream, "[INFO ]"); \
                                break; \
                              case LOG_DEBUG: \
                                dbgstream = stdout; \
                                fprintf(dbgstream, "[DEBUG]"); \
                                break; \
                            } \
                            fprintf(dbgstream, "[%-5d]", (int) getpid()); \
                            snprintf(where, 35, "%s:%d", __FILE__, __LINE__); \
                            fprintf(dbgstream, " %-25s ", where); \
                            fprintf(dbgstream, __VA_ARGS__); \
                            fprintf(dbgstream, "\n"); \
                            fflush(dbgstream); \
                          } \
                        } while (0)


#endif
```

7.19 messages.cpp File Reference

Implementation of messages.h.

```
#include <cstdlib>
#include "network/messages.h"
#include "utils/dump_buffer.h"
```

Functions

- Message ∗ readMessage (char ∗buffer, int len)

  Reads the message using the correct class and returns a pointer to it.

7.19.1 Detailed Description

Implementation of messages.h.

Author

Riccardo Mancini

See also

messages.h

Definition in file messages.cpp.

7.19.2 Function Documentation

7.19.2.1 readMessage()

```
Message* readMessage (
    char * buffer,
    int len )
```

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created Message when you are done with it.

Definition at line 15 of file messages.cpp.

7.20 messages.cpp

```
#include <cstdlib>

#include "network/messages.h"
#include "utils/dump_buffer.h"

Message* readMessage(char *buffer, int len){
    Message *m;
    int ret;

    switch(buffer[0]){
        case START_GAME:
            m = new StartGameMessage;
            break;
        case MOVE:
            m = new MoveMessage;
            break;
        default:
            m = NULL;
            LOG(LOG_ERR, "Unrecognized message type %d", buffer[0]);
            dump_buffer_hex(buffer, len);
            return NULL;
    };

    ret = m->read(buffer, len);

    if (ret != 0){
        LOG(LOG_ERR, "Error reading message of type %d: %d", buffer[0], ret);
        dump_buffer_hex(buffer, len);
        return NULL;
    } else{
        return m;
    }
}


int StartGameMessage::write(char *buffer){
    buffer[0] = (char) START_GAME;
    return 1;
}

int StartGameMessage::read(char *buffer, int len){
    return 0;
}

int MoveMessage::write(char *buffer){
    buffer[0] = (char) MOVE;
    buffer[1] = col;
    return 2;
}

int MoveMessage::read(char *buffer, int len){
    if (len < 2)
        return 1;

    col = buffer[1];
    return 0;
}
```

7.21 messages.h File Reference

Definition of messages.

```
#include "logging.h"
#include <string>
#include "network/inet_utils.h"
```

Data Structures

- class Message

  Abstract class for Messages.
- class StartGameMessage

Message that signals to start a new game.

- class MoveMessage

  Message that signals a move.

Macros

- #define MAX_MSG_SIZE 1024

  Maximum message size.

Enumerations

- enum MessageType { **START_GAME**, **MOVE** }

  Possible type of messages.

Functions

- Message ∗ readMessage (char ∗buffer, int len)

  Reads the message using the correct class and returns a pointer to it.

7.21.1 Detailed Description

Definition of messages.

Author

Riccardo Mancini

Date

2020-05-17

Definition in file messages.h.

7.21.2 Macro Definition Documentation

7.21.2.1 MAX_MSG_SIZE `#define MAX_MSG_SIZE 1024`

Maximum message size.

TODO: it is random, calculate it

Definition at line 24 of file messages.h.

7.21.3 Enumeration Type Documentation

7.21.3.1 MessageType `enum MessageType`

Possible type of messages.

When adding a new message class, add a related type here and set its getType method to return it.

Definition at line 32 of file messages.h.

7.21.4 Function Documentation

7.21.4.1 readMessage()

```
Message* readMessage (
        char * buffer,
        int len )
```

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created Message when you are done with it.

Definition at line 15 of file messages.cpp.

7.22 messages.h

```
00001
00010 #ifndef MESSAGES_H
00011 #define MESSAGES_H
00012
00013 #include "logging.h"
00014 #include <string>
00015 #include "network/inet_utils.h"
00016
00017 using namespace std;
00018
00024 #define MAX_MSG_SIZE 1024
00025
00032 enum MessageType {START_GAME, MOVE};
00033
00037 class Message{
00038 public:
00044     virtual int write(char *buffer) = 0;
00045
00052     virtual int read(char *buffer, int len) = 0;
00053
00057     virtual size_t size() = 0;
00058
00062     virtual string getName() = 0;
00063
00064     virtual MessageType getType() = 0;
00065 };
00066
00070 class StartGameMessage : public Message{
00071 public:
00072     StartGameMessage() {}
00073
00074     int write(char *buffer);
00075     int read(char *buffer, int len);
00076
00077     size_t size(){return 1;}
00078
00079     string getName(){return "StartGame";}
```

```

    MessageType getType(){return START_GAME;}
};

class MoveMessage : public Message{
private:
    char col;
public:
    MoveMessage(){}
    MoveMessage(char col) : col(col) {}

    int write(char *buffer);
    int read(char *buffer, int len);

    size_t size(){return 2;}

    string getName(){return "Move";}

    char getColumn(){return col;}

    MessageType getType(){return MOVE;}
};

Message* readMessage(char *buffer, int len);

#endif // MESSAGES_H
```

7.23 single_player.cpp File Reference

Implementation of a 4-in-a-row game.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"
```

Functions

- void **printWelcome** ()
- int **main** ()

Variables

- char **players** [ ] = {'X', 'O'}

7.23.1 Detailed Description

Implementation of a 4-in-a-row game.

Author

Mirko Laruina

Date

2020-05-17

Definition in file single_player.cpp.

7.24 single_player.cpp

```cpp

#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"

using namespace std;

char players[] = {'X', 'O'};

void printWelcome(){
    cout«"***************************************************************\n"
        «"*        __ __     _                                           *\n"
        «"*       / // /    (_)___     ____ _   _________ _      __      *\n"
        «"*      / // /_   / / __ \\    / __ `/  / ___/ __ \\ | /| / /       *\n"
        «"*     /__  __/  / / / / /  / /_/ /  / /  / /_/ / |/ |/ /       *\n"
        «"*       /_/    /_/_/ /_/   \\__,_/  /_/   \\____/|__/|__/        *\n"
        «"*                                                              *\n"
        «"***************************************************************"
        «endl;
}

int main(){
    // handle server selection on params here
    char in_buffer[256];
    int choosen_col;
    int adv_col;
    int win;
    srand(time(NULL));

    printWelcome();
    cout«endl«"Welcome to 4-in-a-row!"«endl;
    cout«"The rules of the game are simple: you win when you have 4 connected tokens along any direction."«endl;

    Connect4 c;
    cout«"Who do you want to be? X or O ?"«endl;

    do {
        cout«"> ";
        cin.getline(in_buffer, sizeof(in_buffer));
    } while (!c.setPlayer(in_buffer[0]));
    cout«"You are playing as "«c.getPlayer()«endl;
    cout«"This is the starting board:"«endl;
    cout«c;

    do {
        cout«"Write the column you want to insert the token to"«endl;
        do {
            cout«"> ";
            cin.getline(in_buffer, sizeof(in_buffer));
            choosen_col = in_buffer[0]-'0';
        } while(choosen_col < 0 || choosen_col > 7);

        win = c.play(choosen_col-1, c.getPlayer());
        cout«c;
        if(win == 1){
            cout«"Congratulation, you won!"«endl;
        } else if(win == -1){
            cout«"The column is full, choose a different one!"«endl;
            continue;
        } else if(win == -2){
            cout«"The entire board is filled: it is a draw!"«endl;
            break;
        }

        if(win != 1){
            do {
                adv_col = rand()%c.getNumCols();
                cout«"Your enemy has chosen column "«adv_col«endl;
                win = c.play(adv_col, c.getAdv());
                cout«c;
                if(win == 1){
                    cout«"Damn! You lost!"«endl;
                } else if(win == -1){
                    cout«"The column is full, the adversary has to chose a different one!"«endl;
                    continue;
                } else if(win == -2){
                    cout«"The entire board is filled: it is a draw!"«endl;
                    break;
                }
            } while (win == -1);
        }
    } while (win == -1 || win == 0);
    return 0;
}
```

7.25 socket_wrapper.cpp File Reference

Implementation of socket_wrapper.h.

```
#include "logging.h"
#include "network/socket_wrapper.h"
```

7.25.1 Detailed Description

Implementation of socket_wrapper.h.

Author

Riccardo Mancini

See also

socket_wrapper.h

Definition in file socket_wrapper.cpp.

7.26 socket_wrapper.cpp

```

#include "logging.h"
#include "network/socket_wrapper.h"

SocketWrapper::SocketWrapper() {
    socket_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_fd < 0){
        LOG(LOG_ERR, "Could not create socket!\n");
        perror("Error: ");
        return;
    }
}

Message* SocketWrapper::receiveAnyMsg(size_t size){
    int len;
    char* in_buffer;

    in_buffer = (char*) malloc(size);

    len = recv(socket_fd, in_buffer, size, 0);

    Message *m = readMessage(in_buffer, len);

    free(in_buffer);

    return m;
}

Message* SocketWrapper::receiveMsg(MessageType type, size_t size /*=MAX_MSG_SIZE*/){
    return this->receiveMsg(&type, 1, size);
}

Message* SocketWrapper::receiveMsg(MessageType type[], int n_types,
                                    size_t size /*=MAX_MSG_SIZE*/){
    Message *m = NULL;
    while (m == NULL){
        m = this->receiveAnyMsg();
        if (m != NULL){
            for (int i = 0; i < n_types; i++){
                if (m->getType() == type[i]){
                    return m;
                }
                LOG(LOG_WARN, "Received unexpected message of type %d",  m->getType());
            }
        }
    }
    //TODO: add timeout?
    return NULL;
}


int SocketWrapper::sendMsg(Message *msg){
    int msglen, len;
    char *out_buffer;

    msglen = msg->size();
    out_buffer = (char*) malloc(msglen);

    msg->write(out_buffer);

    len = send(socket_fd, out_buffer, msglen, 0);
    if (len != msglen){
        LOG(LOG_ERR, "Error sending %s: len (%d) != msglen (%d)",
            msg->getName().c_str(),
            len,
            msglen
        );
        return 1;
    }

    LOG(LOG_DEBUG, "Sent message %s", msg->getName().c_str());

    free(out_buffer);
    return 0;
}

int ClientSocketWrapper::connectServer(Host host){
    int ret;

    other_addr = host.getAddress();

    ret = connect(
        socket_fd,
        (struct sockaddr*) &other_addr,
        sizeof(other_addr)
    );

    if (ret != 0){
        LOG(LOG_ERR, "Error connecting to %s",
            sockaddr_in_to_string(host.getAddress()).c_str());
        perror("Error: ");
        return ret;
    }

    return ret;
}

ServerSocketWrapper::ServerSocketWrapper(){
    my_addr = make_my_sockaddr_in(0);
    int ret = bind_random_port(socket_fd, &my_addr);
    if (ret <= 0){
        LOG(LOG_ERR, "Error in binding\n");
        perror("Error: ");
    }

    ret = listen(socket_fd, 10);
    if (ret != 0){
        LOG(LOG_ERR, "Error in setting socket to listen mode\n");
        perror("Error: ");
    }
}

ServerSocketWrapper::ServerSocketWrapper(int port){
    my_addr = make_my_sockaddr_in(port);
    int ret = bind(socket_fd, (struct sockaddr*) &my_addr, sizeof(my_addr));
    if (ret != 0){
        LOG(LOG_ERR, "Error in binding\n");
        perror("Error: ");
    }

    ret = listen(socket_fd, 10);
    if (ret != 0){
        LOG(LOG_ERR, "Error in setting socket to listen mode\n");
        perror("Error: ");
    }
}
```

```

SocketWrapper* ServerSocketWrapper::acceptClient(){
    socklen_t len = sizeof(other_addr);
    int new_sd = accept(
        socket_fd,
        (struct sockaddr*) &other_addr,
        &len
    );

    SocketWrapper *sw = new SocketWrapper(new_sd);
    sw->setOtherAddr(other_addr);
    return sw;
}
```

7.27 socket_wrapper.h File Reference

Definition of the helper class "SocketWrapper" and derivatives.

```
#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"
#include "network/host.h"
```

Data Structures

- class SocketWrapper

  Wrapper class around sockaddr_in and socket descriptor.
- class ClientSocketWrapper

  SocketWrapper for a TCP client.
- class ServerSocketWrapper

  SocketWrapper for a TCP server.

7.27.1 Detailed Description

Definition of the helper class "SocketWrapper" and derivatives.

Author

Riccardo Mancini

Date

2020-05-17

Definition in file socket_wrapper.h.

7.28 socket_wrapper.h

```

#ifndef SOCKET_WRAPPER_H
#define SOCKET_WRAPPER_H

#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"
#include "network/host.h"

class SocketWrapper{
protected:
    struct sockaddr_in other_addr;

    int socket_fd;
public:
    SocketWrapper();

    SocketWrapper(int sd) : socket_fd(sd) {}

    int getDescriptor(){return socket_fd;};

    Message* receiveAnyMsg(size_t size=MAX_MSG_SIZE);

    Message* receiveMsg(MessageType type, size_t size=MAX_MSG_SIZE);

    Message* receiveMsg(MessageType type[], int n_types, size_t size=MAX_MSG_SIZE);

    int sendMsg(Message *msg);

    void setOtherAddr(struct sockaddr_in addr){other_addr = addr;}

    Host getConnectedHost(){return Host(other_addr);}
};

class ClientSocketWrapper : public SocketWrapper{
public:
    int connectServer(Host host);
};

class ServerSocketWrapper : public SocketWrapper{
private:
    struct sockaddr_in my_addr;
public:
    ServerSocketWrapper();

    ServerSocketWrapper(int port);

    SocketWrapper* acceptClient();

    int getPort(){return ntohs(my_addr.sin_port);}
};

#endif // SOCKET_WRAPPER_H
```

Index