Laboratorio_Gr3

Generated by Doxygen 1.8.13

Contents

Chapter 1

Lab3_gr3

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

participante . . . . . ??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

CMakeFiles/3.25.0/CompilerIdC/CMakeCCompilerId.c . . . . . ??
CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp . . . . . ??
CMakeFiles/sofware1.bin.dir/src/config_tty.cpp.o.d . . . . . ??
CMakeFiles/sofware1.bin.dir/src/main.cpp.o.d . . . . . ??
CMakeFiles/sofware1.bin.dir/src/participant.cpp.o.d . . . . . ??
include/lib_grupo3.h . . . . . ??
include/tty_lib2.h . . . . . ??
src/config_tty.cpp . . . . . ??
src/main.cpp . . . . . ??
src/participant.cpp . . . . . ??

Chapter 4

Class Documentation

4.1 participante Class Reference

`#include <lib_grupo3.h>`

Public Member Functions

- participante ()
- participante (unsigned int id, string nom)
- void set_participant (unsigned int id, string nom)
- void set_pushed (unsigned int cant)
- unsigned int get_participant_id ()
- string get_nombre ()
- unsigned int get_pushed ()

4.1.1 Detailed Description

Definition at line 12 of file lib_grupo3.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 participante() [1/2]

`participante::participante ( )`

Definition at line 6 of file participant.cpp.

```
6 : participant_id(0), nombre ("") {}
```

4.1.2.2 participante() [2/2]

```
participante::participante (
            unsigned int id,
            string nom )
```

Definition at line 7 of file participant.cpp.

```
7 : participant_id (id),nombre(nom){}
```

4.1.3 Member Function Documentation

4.1.3.1 get_nombre()

```
string participante::get_nombre ( )
```

Definition at line 22 of file participant.cpp.

```
23 {
24         return nombre;
25 }
```

4.1.3.2 get_participant_id()

```
unsigned int participante::get_participant_id ( )
```

Definition at line 18 of file participant.cpp.

```
19 {
20         return participant_id;
21 }
```

4.1.3.3 get_pushed()

```
unsigned int participante::get_pushed ( )
```

Definition at line 26 of file participant.cpp.

```
27 {
28         return veces_pushed;
29 }
```

4.1.3.4 set_participant()

```
void participante::set_participant (
             unsigned int id,
             string nom )
```

Definition at line 9 of file participant.cpp.

```
10 {
11         participant_id = id;
12         nombre = nom;
13 }
```

4.1.3.5 set_pushed()

```
void participante::set_pushed (
             unsigned int cant )
```

Definition at line 14 of file participant.cpp.

```
15 {
16         veces_pushed=cant;
17 }
```

The documentation for this class was generated from the following files:

- include/lib_grupo3.h
- src/participant.cpp

Chapter 5

File Documentation

5.1 CMakeFiles/3.25.0/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

- int main (int argc, char *argv[ ])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

5.1.1 Macro Definition Documentation

5.1.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

5.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 718 of file CMakeCCompilerId.c.

5.1.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 807 of file CMakeCCompilerId.c.

5.1.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 429 of file CMakeCCompilerId.c.

5.1.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 722 of file CMakeCCompilerId.c.

5.1.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 733 of file CMakeCCompilerId.c.

5.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 560 of file CMakeCCompilerId.c.

5.1.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 450 of file CMakeCCompilerId.c.

5.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 449 of file CMakeCCompilerId.c.

5.1.2 Function Documentation

5.1.2.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 841 of file CMakeCCompilerId.c.

```
843 {
844   int require = 0;
845   require += info_compiler[argc];
846   require += info_platform[argc];
847   require += info_arch[argc];
848 #ifdef COMPILER_VERSION_MAJOR
849   require += info_version[argc];
850 #endif
851 #ifdef COMPILER_VERSION_INTERNAL
852   require += info_version_internal[argc];
853 #endif
854 #ifdef SIMULATE_ID
855   require += info_simulate[argc];
856 #endif
857 #ifdef SIMULATE_VERSION_MAJOR
858   require += info_simulate_version[argc];
859 #endif
860 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
861   require += info_cray[argc];
862 #endif
863   require += info_language_standard_default[argc];
864   require += info_language_extensions_default[argc];
865   (void)argv;
866   return require;
867 }
```

5.1.3 Variable Documentation

5.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 799 of file CMakeCCompilerId.c.

5.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 436 of file CMakeCCompilerId.c.

5.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["



  "OFF"
"]"
```

Definition at line 823 of file CMakeCCompilerId.c.

5.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 820 of file CMakeCCompilerId.c.

5.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 798 of file CMakeCCompilerId.c.

5.2 CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_standard_default
- const char ∗ info_language_extensions_default

5.2.1 Macro Definition Documentation

5.2.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

5.2.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 703 of file CMakeCXXCompilerId.cpp.

5.2.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 414 of file CMakeCXXCompilerId.cpp.

5.2.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 801 of file CMakeCXXCompilerId.cpp.

5.2.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 707 of file CMakeCXXCompilerId.cpp.

5.2.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

5.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 545 of file CMakeCXXCompilerId.cpp.

5.2.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 435 of file CMakeCXXCompilerId.cpp.

5.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

5.2.2 Function Documentation

5.2.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 832 of file CMakeCXXCompilerId.cpp.

```
833 {
834   int require = 0;
835   require += info_compiler[argc];
836   require += info_platform[argc];
837   require += info_arch[argc];
838 #ifdef COMPILER_VERSION_MAJOR
839   require += info_version[argc];
840 #endif
841 #ifdef COMPILER_VERSION_INTERNAL
842   require += info_version_internal[argc];
843 #endif
844 #ifdef SIMULATE_ID
845   require += info_simulate[argc];
846 #endif
847 #ifdef SIMULATE_VERSION_MAJOR
848   require += info_simulate_version[argc];
849 #endif
850 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
851   require += info_cray[argc];
852 #endif
853   require += info_language_standard_default[argc];
854   require += info_language_extensions_default[argc];
855   (void)argv;
856   return require;
857 }
```

5.2.3 Variable Documentation

5.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 784 of file CMakeCXXCompilerId.cpp.

5.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 421 of file CMakeCXXCompilerId.cpp.

5.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["



  "OFF"
"]"
```

Definition at line 820 of file CMakeCXXCompilerId.cpp.

5.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["







  "98"
"]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

5.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 783 of file CMakeCXXCompilerId.cpp.

5.3 CMakeFiles/sofware1.bin.dir/src/config_tty.cpp.o.d File Reference

5.4 CMakeFiles/sofware1.bin.dir/src/main.cpp.o.d File Reference

5.5 CMakeFiles/sofware1.bin.dir/src/participant.cpp.o.d File Reference

5.6 include/lib_grupo3.h File Reference

```
#include <unistd.h>
#include <iostream>
#include <stdlib.h>
```

Classes

- class participante

Functions

- int comunicacion_serial (float t1, float t2)

5.6.1 Function Documentation

5.6.1.1 comunicacion_serial()

```
int comunicacion_serial (
            float t1,
            float t2 )
```

Definition at line 33 of file participant.cpp.

```
34 {
35     struct termios tty;
36         int serial_port;
37     config_tty ("/dev/ttyS0", &tty, B9600, &serial_port);
38     int read_buf;
39         int num_bytes;
40 
41     sleep (2);
42     cout<<"\n Preparados ..."<<endl;
43         write(serial_port, "s", sizeof(char));
44         sleep(t1);
45 
46     cout<<"\n Comience a pulsar"<<endl;
47 
48     write(serial_port, "r", sizeof(char));
49     sleep(t2);
50 
51     cout<<"\n Acabo el tiempo."<<endl;
52     write(serial_port, "S", sizeof(char));
53 
54     num_bytes = read(serial_port, &read_buf, sizeof(read_buf));
55     close(serial_port);
56     return read_buf;
57 }
```

5.7 include/tty_lib2.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
```

Functions

- void config_tty (const char *tty_port, struct termios *tty, unsigned int baud, int *serial_port)

5.7.1 Function Documentation

5.7.1.1 config_tty()

```
void config_tty (
            const char * tty_port,
            struct termios * tty,
            unsigned int baud,
            int * serial_port )
```

Definition at line 8 of file config_tty.cpp.

```
9 {
10
11     *serial_port = open(tty_port, O_RDWR);
12
13     // Check for errors
14     if (*serial_port < 0) {
15         printf("Error %i from open: %s\n", errno, strerror(errno));
16     }
17
18
19     // Create new termios struct, we call it 'tty' for convention
20     // No need for "= {0}" at the end as we'll immediately write the existing
21     // config to this struct
22     //struct termios tty;//no needed here as is received in function argument
23
24     // Read in existing settings, and handle any error
25     // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
26     // must have been initialized with a call to tcgetattr() overwise behaviour
27     // is undefined
28     if(tcgetattr(*serial_port, tty) != 0) {
29         printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
30     }
31
32     tty->c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
33     //tty->c_cflag |= PARENB;  // Set parity bit, enabling parity
34
35     tty->c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
36     //tty->c_cflag |= CSTOPB;  // Set stop field, two stop bits used in communication
37
38
39     tty->c_cflag &= ~CSIZE; // Clear all the size bits, then use one of the statements below
40     //tty->c_cflag |= CS5; // 5 bits
41     //tty->c_cflag |= CS6; // 6 bits
42     //tty->c_cflag |= CS7; // 7 bits
43     tty->c_cflag |= CS8; // 8 bits (most common)
44
```

```
45
46      tty->c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
47      //tty->c_cflag |= CRTSCTS;  // Enable RTS/CTS hardware flow control
48
49      tty->c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
50
51      //In canonical mode, input is processed when a new line character is received.
52      tty->c_lflag &= ~ICANON; // non-canonical
53
54      //If this bit is set, sent characters will be echoed back.
55      tty->c_lflag &= ~ECHO; // Disable echo
56      tty->c_lflag &= ~ECHOE; // Disable erasure
57      tty->c_lflag &= ~ECHONL; // Disable new-line echo
58
59      tty->c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
60
61      tty->c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
62
63      tty->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of
        received bytes
64
65      tty->c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
66      tty->c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
67      // tty->c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT IN LINUX)
68      // tty->c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT IN LINUX)
69
70
71    /*VMIN = 0, VTIME = 0: No blocking, return immediately with what is available
72      VMIN > 0, VTIME = 0: This will make read() always wait for bytes (exactly how many is determined by
        VMIN), so read() could block indefinitely.
73      VMIN = 0, VTIME > 0: This is a blocking read of any number of chars with a maximum timeout (given by
        VTIME). read() will block until either any amount of data is available, or the timeout occurs. This happens to
        be my favourite mode (and the one I use the most).
74      VMIN > 0, VTIME > 0: Block until either VMIN characters have been received, or VTIME after first
        character has elapsed. Note that the timeout for VTIME does not begin until the first character is received.
75      type of VMIN and VTIME: cc_t (1B)*/
76      tty->c_cc[VTIME] = 0;
77      tty->c_cc[VMIN] = 1; // wait one byte
78
79      //B0,  B50,  B75,  B110,  B134,  B150,  B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200,
        B38400, B57600, B115200, B230400, B460800
80      // Set in/out baud rate to be 9600
81      cfsetispeed(tty, baud);
82      cfsetospeed(tty, baud);
83      //cfsetspeed(tty, B9600); //set both input and output
84
85      //cfsetispeed(tty, 104560); //Specifying a custom baud rate when using GNU C
86      //cfsetospeed(tty, 104560);
87
88      /*Other option for custom baud rate*/
89      /*
90          // #include <termios.h> This must be removed!
91          // Otherwise we'll get "redefinition of struct termios" errors
92          #include <sys/ioctl.h> // Used for TCGETS2/TCSETS2, which is required for custom baud rates
93          struct termios2 tty;
94          // Read in the terminal settings using ioctl instead
95          // of tcsetattr (tcsetattr only works with termios, not termios2)
96          ioctl(fd, TCGETS2, tty);
97          // Set everything but baud rate as usual
98          // ...
99          // ...
100
101          // Set custom baud rate
102          tty->c_cflag &= ~CBAUD;
103          tty->c_cflag |= CBAUDEX;
104          // On the internet there is also talk of using the "BOTHER" macro here:
105          // tty->c_cflag |= BOTHER;
106          // I never had any luck with it, so omitting in favour of using
107          // CBAUDEX
108          tty->c_ispeed = 123456; // What a custom baud rate!
109          tty->c_ospeed = 123456;
110
111          // Write terminal settings to file descriptor
112          ioctl(*serial_port, TCSETS2, tty);
113      */
114
115      // Save tty settings, also checking for error
116      if (tcsetattr(*serial_port, TCSANOW, tty) != 0) {
117          printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
118      }
119      /*********/
120      /*WRITING*/
121      /*********/
122      //unsigned char msg[] = { 'H', 'e', 'l', 'l', 'o', '\r' };
123      //write(*serial_port, msg, sizeof(msg));
124
125      /*********/
```

```
126     /*READING*/
127     /*********/
128       // Allocate memory for read buffer, set size according to your needs
129     //char read_buf [256];
130
131     // Normally you wouldn't do this memset() call, but since we will just receive
132     // ASCII data for this example, we'll set everything to 0 so we can
133     // call printf() easily.
134     //memset(&read_buf, '\0', sizeof(read_buf));
135
136     // Read bytes. The behaviour of read() (e.g. does it block?,
137     // how long does it block for?) depends on the configuration
138     // settings above, specifically VMIN and VTIME
139     //int num_bytes = read(*serial_port, &read_buf, sizeof(read_buf));
140
141     // n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1 to signal an
      error.
142     //if (num_bytes < 0) {
143     //  printf("Error reading: %s", strerror(errno));
144     //  return 1;
145     //}
146
147     // Here we assume we received ASCII data, but you might be sending raw bytes (in that case, don't try
      and
148     // print it to the screen like this!)
149     //printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
150
151     //close(serial_port);
152
153 }
```

5.8 README.md File Reference

5.9 src/config_tty.cpp File Reference

```
#include "tty_lib2.h"
```

Functions

- void config_tty (const char *tty_port, struct termios *tty, unsigned int baud, int *serial_port)

5.9.1 Function Documentation

5.9.1.1 config_tty()

```
void config_tty (
            const char * tty_port,
            struct termios * tty,
            unsigned int baud,
            int * serial_port )
```

Definition at line 8 of file config_tty.cpp.

```
9 {
10
11      *serial_port = open(tty_port, O_RDWR);
12
13      // Check for errors
14      if (*serial_port < 0) {
15          printf("Error %i from open: %s\n", errno, strerror(errno));
16      }
17
18
19      // Create new termios struct, we call it 'tty' for convention
20      // No need for "= {0}" at the end as we'll immediately write the existing
21      // config to this struct
22      //struct termios tty;//no needed here as is received in function argument
23
24      // Read in existing settings, and handle any error
25      // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
26      // must have been initialized with a call to tcgetattr() overwise behaviour
27      // is undefined
28      if(tcgetattr(*serial_port, tty) != 0) {
29          printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
30      }
31
32      tty->c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
33      //tty->c_cflag |= PARENB;  // Set parity bit, enabling parity
34
35      tty->c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
36      //tty->c_cflag |= CSTOPB;  // Set stop field, two stop bits used in communication
37
38
39      tty->c_cflag &= ~CSIZE; // Clear all the size bits, then use one of the statements below
40      //tty->c_cflag |= CS5; // 5 bits
41      //tty->c_cflag |= CS6; // 6 bits
42      //tty->c_cflag |= CS7; // 7 bits
43      tty->c_cflag |= CS8; // 8 bits (most common)
44
45
46      tty->c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
47      //tty->c_cflag |= CRTSCTS;  // Enable RTS/CTS hardware flow control
48
49      tty->c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
50
51      //In canonical mode, input is processed when a new line character is received.
52      tty->c_lflag &= ~ICANON; // non-canonical
53
54      //If this bit is set, sent characters will be echoed back.
55      tty->c_lflag &= ~ECHO; // Disable echo
56      tty->c_lflag &= ~ECHOE; // Disable erasure
57      tty->c_lflag &= ~ECHONL; // Disable new-line echo
58
59      tty->c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
60
61      tty->c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
62
63      tty->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of
        received bytes
64
65      tty->c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
66      tty->c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
67      // tty->c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT IN LINUX)
68      // tty->c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT IN LINUX)
69
70
71    /*VMIN = 0, VTIME = 0: No blocking, return immediately with what is available
72      VMIN > 0, VTIME = 0: This will make read() always wait for bytes (exactly how many is determined by
        VMIN), so read() could block indefinitely.
73      VMIN = 0, VTIME > 0: This is a blocking read of any number of chars with a maximum timeout (given by
        VTIME). read() will block until either any amount of data is available, or the timeout occurs. This happens to
        be my favourite mode (and the one I use the most).
74      VMIN > 0, VTIME > 0: Block until either VMIN characters have been received, or VTIME after first
        character has elapsed. Note that the timeout for VTIME does not begin until the first character is received.
75      type of VMIN and VTIME: cc_t (1B)*/
76      tty->c_cc[VTIME] = 0;
77      tty->c_cc[VMIN] = 1; // wait one byte
78
79      //B0,  B50,  B75,  B110,  B134,  B150,  B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200,
        B38400, B57600, B115200, B230400, B460800
80      // Set in/out baud rate to be 9600
81      cfsetispeed(tty, baud);
82      cfsetospeed(tty, baud);
83      //cfsetspeed(tty, B9600); //set both input and output
84
85      //cfsetispeed(tty, 104560); //Specifying a custom baud rate when using GNU C
86      //cfsetospeed(tty, 104560);
87
88      /*Other option for custom baud rate*/
89      /*
```

```
90          // #include <termios.h> This must be removed!
91          // Otherwise we'll get "redefinition of struct termios" errors
92          #include <sys/ioctl.h> // Used for TCGETS2/TCSETS2, which is required for custom baud rates
93          struct termios2 tty;
94          // Read in the terminal settings using ioctl instead
95          // of tcsetattr (tcsetattr only works with termios, not termios2)
96          ioctl(fd, TCGETS2, tty);
97          // Set everything but baud rate as usual
98          // ...
99          // ...
100
101          // Set custom baud rate
102          tty->c_cflag &= ~CBAUD;
103          tty->c_cflag |= CBAUDEX;
104          // On the internet there is also talk of using the "BOTHER" macro here:
105          // tty->c_cflag |= BOTHER;
106          // I never had any luck with it, so omitting in favour of using
107          // CBAUDEX
108          tty->c_ispeed = 123456; // What a custom baud rate!
109          tty->c_ospeed = 123456;
110
111          // Write terminal settings to file descriptor
112          ioctl(*serial_port, TCSETS2, tty);
113      */
114
115      // Save tty settings, also checking for error
116      if (tcsetattr(*serial_port, TCSANOW, tty) != 0) {
117          printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
118      }
119      /*********/
120      /*WRITING*/
121      /*********/
122      //unsigned char msg[] = { 'H', 'e', 'l', 'l', 'o', '\r' };
123      //write(*serial_port, msg, sizeof(msg));
124
125      /*********/
126      /*READING*/
127      /*********/
128        // Allocate memory for read buffer, set size according to your needs
129      //char read_buf [256];
130
131      // Normally you wouldn't do this memset() call, but since we will just receive
132      // ASCII data for this example, we'll set everything to 0 so we can
133      // call printf() easily.
134      //memset(&read_buf, '\0', sizeof(read_buf));
135
136      // Read bytes. The behaviour of read() (e.g. does it block?,
137      // how long does it block for?) depends on the configuration
138      // settings above, specifically VMIN and VTIME
139      //int num_bytes = read(*serial_port, &read_buf, sizeof(read_buf));
140
141      // n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1 to signal an
     error.
142      //if (num_bytes < 0) {
143      //  printf("Error reading: %s", strerror(errno));
144      //  return 1;
145      //}
146
147      // Here we assume we received ASCII data, but you might be sending raw bytes (in that case, don't try
     and
148      // print it to the screen like this!)
149      //printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
150
151      //close(serial_port);
152
153  }
```

5.10 src/main.cpp File Reference

```
#include "tty_lib2.h"
#include "lib_grupo3.h"
```

Functions

- int main (int argc, char *argv[ ])

5.10.1 Function Documentation

5.10.1.1 main()

```
int main (
              int argc,
              char * argv[] )
```

Definition at line 4 of file main.cpp.

```
5 {
6         float time_pre, time_venta;
7     int max=2; //cantidad de participantes
8         int aux=0; // auxiliar para el ciclo
9     int read_buf;
10     int mayor=0;
11     int ganador;
12         string  nom;
13         unsigned int id=0;
14
15         if (argc !=3)
16         {
17                 cout <<"Error en la cantidad de argumentos"<<endl;
18         }
19
20         else
21         {
22         time_pre=atof(argv[1]);
23                 time_venta=atof(argv[2]);
24
25     }
26
27         cout<<"\n\n Ingrese la cantidad de participantes:";
28         cin >> max ;
29         participante participantes[max];
30     for (aux=0; aux<max ;aux++)
31         {
32                 id=0;
33                 cout<<"\n Nombre del participante "<<aux+1<<":";
34                 cin>>nom;
35                 cout<<"\n Número de identiticación:";
36                 cin>>id;
37                 participantes[aux].set_participant (id,nom);
38
39         read_buf= comunicacion_serial(time_pre,time_venta);
40
41         participantes[aux].set_pushed (char(read_buf));
42
43         }
44     for (aux=0; aux<max ;aux++)
45     {
46         if (mayor<participantes[aux].get_pushed ())
47         {
48             mayor=participantes[aux].get_pushed ();
49             ganador=aux;
50         }
51
52     }
53     cout << "\n \t \t \t RESULTADOS:"<<endl;
54     cout << "Participante \t\t Número de identiticación \t\t Cantidad de pulsos:"<< endl;
55     for (aux=0; aux<max ;aux++)
56         {
57         //cout << "Participante " <<aux+1 << ":" <<participantes[aux].get_nombre ()<< endl;
58         //cout << "Número de identiticación:" <<participantes[aux].get_participant_id ()<< endl;
59         //cout << "Cantidad de pulsos:" <<participantes[aux].get_pushed ()<< endl;
60
61         cout <<participantes[aux].get_nombre ();
62         cout <<"\t\t\t\t"<<participantes[aux].get_participant_id ();
63         cout <<"\t\t\t\t\t"<<participantes[aux].get_pushed ()<< endl;
64
65     }
66     cout<<"\n El ganador es "<<participantes[ganador].get_nombre ();
67     cout<<" con el siguiete puntaje "<< mayor << endl;
68
69 }
```

5.11 src/participant.cpp File Reference

```
#include "tty_lib2.h"
#include "lib_grupo3.h"
```

Functions

- int comunicacion_serial (float t1, float t2)

5.11.1 Function Documentation

5.11.1.1 comunicacion_serial()

```
int comunicacion_serial (
            float t1,
            float t2 )
```

Definition at line 33 of file participant.cpp.

```
34 {
35     struct termios tty;
36         int serial_port;
37     config_tty ("/dev/ttyS0", &tty, B9600, &serial_port);
38     int read_buf;
39         int num_bytes;
40 
41     sleep (2);
42     cout<<"\n Preparados ..."<<endl;
43         write(serial_port, "s", sizeof(char));
44         sleep(t1);
45 
46     cout<<"\n Comience a pulsar"<<endl;
47 
48     write(serial_port, "r", sizeof(char));
49     sleep(t2);
50 
51     cout<<"\n Acabo el tiempo."<<endl;
52     write(serial_port, "S", sizeof(char));
53 
54     num_bytes = read(serial_port, &read_buf, sizeof(read_buf));
55     close(serial_port);
56     return read_buf;
57 }
```