0pt

Hofmeier_FYP:libcomm Reference Manual

Generated by Doxygen 1.2.15

Tue Mar 29 21:05:25 2005

Contents

Chapter 1

Hofmeier_FYP:libcomm Data Structure Index

1.1 Hofmeier_FYP:libcomm Data Structures

Here are the data structures with brief descriptions:

Chapter 2

Hofmeier_FYP:libcomm File Index

2.1 Hofmeier_FYP:libcomm File List

Here is a list of all files with brief descriptions:

Chapter 3

Hofmeier_FYP:libcomm Data Structure Documentation

3.1 AUTHINFO Struct Reference

`#include <libcomm.h>`

Data Fields

- char * **netname**
- char * **name**
- char * **passwd**
- char * **keyencrypt**
- char * **keydecrypt**

3.1.1 Field Documentation

3.1.1.1 char* AUTHINFO::keydecrypt

3.1.1.2 char* AUTHINFO::keyencrypt

3.1.1.3 char* AUTHINFO::name

3.1.1.4 char* AUTHINFO::netname

3.1.1.5 char* AUTHINFO::passwd

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

- **libcomm.h**

3.2 LIBCOMMPTHREADP Struct Reference

`#include <libcomm.h>`

Data Fields

- void(* **socket_accept_do**)(int fd, int **id**, char *pip, struct sockaddr_in their_addr)
- pthread_t **thrd_2**
- pthread_attr_t **thrd_2_attr**
- int **sockport**
- int **id**

3.2.1 Field Documentation

3.2.1.1 int LIBCOMMPTHREADP::id

3.2.1.2 void(* LIBCOMMPTHREADP::socket_accept_do)(int fd, int id, char *pip, struct sockaddr_in their_addr)

3.2.1.3 int LIBCOMMPTHREADP::sockport

3.2.1.4 pthread_t LIBCOMMPTHREADP::thrd_2

3.2.1.5 pthread_attr_t LIBCOMMPTHREADP::thrd_2_attr

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

- **libcomm.h**

3.3 LIBCOMMPTHREADS Struct Reference

`#include <libcomm.h>`

Data Fields

- void(* **block_call_do**)(int **fd**, int **id**, unsigned int type, char *buf, unsigned int size, int **term**)
- void(* **block_call_term**)(int **fd**, int **id**)
- pthread_t **thrd_1**
- pthread_attr_t **thrd_1_attr**
- int **fd**
- int **id**
- int **term**

3.3.1 Field Documentation

3.3.1.1 void(* LIBCOMMPTHREADS::block_call_do)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term)

3.3.1.2 void(* LIBCOMMPTHREADS::block_call_term)(int fd, int id)

3.3.1.3 int LIBCOMMPTHREADS::fd

3.3.1.4 int LIBCOMMPTHREADS::id

3.3.1.5 int LIBCOMMPTHREADS::term

3.3.1.6 pthread_t LIBCOMMPTHREADS::thrd_1

3.3.1.7 pthread_attr_t LIBCOMMPTHREADS::thrd_1_attr

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

- **libcomm.h**

3.4 LINEMONITOR_THREAD_DATA Struct Reference

`#include <libcomm.h>`

Data Fields

- char * **server**
- int **port**
- int **soft_msec**
- int **hard_msec**
- int **wait_msec**
- void(* **linemonitor_exception**)(char ***server**, int **port**, int type)
- int **sock**

3.4.1 Field Documentation

3.4.1.1 int LINEMONITOR_THREAD_DATA::hard_msec

3.4.1.2 void(* LINEMONITOR_THREAD_DATA::linemonitor_exception)(char *server, int port, int type)

3.4.1.3 int LINEMONITOR_THREAD_DATA::port

3.4.1.4 char* LINEMONITOR_THREAD_DATA::server

3.4.1.5 int LINEMONITOR_THREAD_DATA::sock

3.4.1.6 int LINEMONITOR_THREAD_DATA::soft_msec

3.4.1.7 int LINEMONITOR_THREAD_DATA::wait_msec

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

- **libcomm.h**

3.5 MD5Context Struct Reference

`#include <md5.h>`

Data Fields

- **uint32 buf** [4]
- **uint32 bits** [2]
- unsigned char **in** [64]

3.5.1 Field Documentation

3.5.1.1 uint32 MD5Context::bits[2]

3.5.1.2 uint32 MD5Context::buf[4]

3.5.1.3 unsigned char MD5Context::in[64]

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

- **md5.h**

3.6 ROBOT_POSITION Struct Reference

`#include <guicommon.h>`

Data Fields

- int **x**
- int **y**

3.6.1 Field Documentation

3.6.1.1 int ROBOT_POSITION::x

3.6.1.2 int ROBOT_POSITION::y

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

- **guicommon.h**

Chapter 4

Hofmeier_FYP:libcomm File Documentation

4.1 fail.c File Reference

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

Functions

- **fail** (char *errmsg, int term)
  This function prints a fail message, waits for an input and may terminates the programm.

4.1.1 Function Documentation

4.1.1.1 fail (char * *errmsg*, int *term*)

This function prints a fail message, waits for an input and may terminates the programm.

Parameters:

errmsg a string which contains the error message.

term a integer. The functions calls exit(1) if this value is not equal to 0

4.2 guicommon.0.1.complmov.c File Reference

```
#include <time.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib.h>
```

Functions

- void **gui_common_drawrobot** (GtkWidget ∗widget, int ∗x, int ∗y, int mx, int my, int mouseaction)

  Function for drawing the robot and to calculate the absolute postion of the robot out of the mouse-position.

4.2.1 Function Documentation

4.2.1.1 void gui_common_drawrobot (GtkWidget ∗ *widget*, int ∗ *x*, int ∗ *y*, int *mx*, int *my*, int *mouseaction*)

Function for drawing the robot and to calculate the absolute postion of the robot out of the mouse-position.

Parameters:

widget (GtkWidget ∗) pointer to affected object (drawing area).

x (int ∗) pointer to absolute x position of the robot (will changed if mouseaction equal to 1 or 2)

y (int ∗) pointer to absolute y position of the robot (will changed if mouseaction equal to 1 or 2)

mx (int) x mouse position (only relevant if mouseaction equal to 1 or 2)

my (int) y mouse position (only relevant if mouseaction equal to 1 or 2)

mouseaction (int) current action: 1: start moving, 2: moving or finish moving, other: plot only

4.3 guicommon.c File Reference

`#include <time.h>`

`#include <math.h>`

`#include <pthread.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <gtk/gtk.h>`

`#include <glib.h>`

Functions

- void **gui_common_drawrobot** (GtkWidget ∗widget, int ∗x, int ∗y, int mx, int my, int mouseaction)

    Function for drawing the robot and to calculate the absolute postion of the robot out of the mouse-position.

4.3.1 Function Documentation

4.3.1.1 void gui_common_drawrobot (GtkWidget ∗ *widget*, int ∗ *x*, int ∗ *y*, int *mx*, int *my*, int *mouseaction*)

Function for drawing the robot and to calculate the absolute postion of the robot out of the mouse-position.

Parameters:

widget (GtkWidget ∗) pointer to affected object (drawing area).

x (int ∗) pointer to absolute x position of the robot (will changed if mouseaction equal to 1 or 2)

y (int ∗) pointer to absolute y position of the robot (will changed if mouseaction equal to 1 or 2)

mx (int) x mouse position (only relevant if mouseaction equal to 1 or 2)

my (int) y mouse position (only relevant if mouseaction equal to 1 or 2)

mouseaction (int) current action: 1: start moving, 2: moving or finish moving, other: plot only

4.4 guicommon.h File Reference

Data Structures

- struct **ROBOT_POSITION**

4.5 guirobot.c File Reference

Robot-Side application with GUI (simulator) to control the robot.

`#include <time.h>`

`#include <math.h>`

`#include <pthread.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <gtk/gtk.h>`

`#include <glib.h>`

`#include <unistd.h>`

`#include <signal.h>`

`#include <gdk/gdk.h>`

`#include "../lib/libcomm.h"`

`#include "guicommon.h"`

Functions

- void **gr_delete_event** (GtkWidget *widget, gpointer data)

  This function will be called be gtk when the window gets a closing command, it makes sure that the interface is down and the program is finished.

- void **gr_da_expose_event** (GtkWidget *widget, gpointer data)

  This function is called by gtk if the drawing area (da) needs to be redrawn.

- void **gr_button_press_event** (GtkWidget *widget, gpointer data)

  This function is called if a button is pressed.

- **gr_block_call_do_test** (int fd, int id, unsigned int type, char *buf, unsigned int size, int term)

  Function which is called from **block_call***()* (p. 28) *if a message/datablock has received.*

- **linemonitor_exception** (char *server, int port, int type)

  Exception function for the **linemonitor***()* (p. 31)*, print exception code and its meaning on the screen and take further action if necessary.*

- **gr_block_call_term_test** (int fd, int id)

  Function which ist called from **block_call***()* (p. 28) *if a the connection terminates: shut down system.*

- **gr_socket_accept_do** (int fd, int id, char *pip, struct sockaddr_in their_addr, int term)

  Function which is called from **socket_accept***()* (p. 32) *if someone has connected.*

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

  test0001sockts: see test0001sockts.c.

Variables

- GtkWidget * **gr_application**
- GtkWidget * **da_global**
- int **robot_x** = 50
- int **robot_y** = 0
- int **soft_msec**
- int **hard_msec**
- int **wait_msec**
- int **shut_down**
- int **serverport**

4.5.1 Detailed Description

Robot-Side application with GUI (simulator) to control the robot.

4.5.2 Function Documentation

4.5.2.1 gr_block_call_do_test (int *fd*, int *id*, unsigned int *type*, char * *buf*, unsigned int *size*, int *term*)

Function which is called from **block_call**() (p. 28) if a message/datablock has received.

See API of **block_call**() (p. 28).

4.5.2.2 gr_block_call_term_test (int *fd*, int *id*)

Function which ist called from **block_call**() (p. 28) if a the connection terminates: shut down system.

See **block_call**() (p. 28) API.

4.5.2.3 void gr_button_press_event (GtkWidget * *widget*, gpointer *data*)

This function is called if a button is pressed.

Parameters:

widget (GtkWidget *) pointer to affected object (button).

event (GdkEventMotion *) pointer to structure which describes the event.

4.5.2.4 void gr_da_expose_event (GtkWidget * *widget*, gpointer *data*)

This function is called by gtk if the drawing area (da) needs to be redrawn.

For example if it becomes visible.

Parameters:

widget (GtkWidget *) pointer to affected object (in this case the drawing area).

data (gpointer) pointer to additional data from gtk.

4.5.2.5 void gr_delete_event (GtkWidget * *widget*, gpointer *data*)

This function will be called be gtk when the window gets a closing command, it makes sure that the interface is down and the program is finished.

Parameters:

widget (GtkWidget *) pointer to affected object (for example: window).

data (gpointer) pointer to additional data from gtk.

4.5.2.6 gr_socket_accept_do (int *fd*, int *id*, char * *pip*, struct sockaddr_in *their_addr*, int *term*)

Function which is called from **socket_accept**() (p. 32) if someone has connected.

See **socket_accept**() (p. 32) API.

4.5.2.7 linemonitor_exception (char * *server*, int *port*, int *type*)

Exception function for the **linemonitor**() (p. 31), print exception code and its meaning on the screen and take further action if necessary.

See API of **linemonitor**() (p. 31).

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

test0001sockts: see test0001sockts.c.

4.5.3 Variable Documentation

4.5.3.1 GtkWidget* da_global

4.5.3.2 GtkWidget* gr_application

4.5.3.3 int hard_msec

4.5.3.4 int robot_x = 50

4.5.3.5 int robot_y = 0

4.5.3.6 int serverport

4.5.3.7 int shut_down

4.5.3.8 int soft_msec

4.5.3.9 int wait_msec

4.6 guiserver.c File Reference

Server/User-Side Application with GUI to control the robot.

```
#include <time.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib.h>
#include <unistd.h>
#include <signal.h>
#include "../lib/libcomm.h"
#include "guicommon.h"
```

Functions

- void **gs_delete_event** (GtkWidget ∗widget, gpointer data)

  This function will be called be gtk when the window gets a closing command, it makes sure that the interface is down and the program is finished.

- void **gs_da_expose_event** (GtkWidget ∗widget, gpointer data)

  This function is called by gtk if the drawing area (da) needs to be redrawn.

- void **gs_button_press_event** (GtkWidget ∗widget, gpointer data)

  This function is called if a button is pressed.

- **linemonitor_exception** (char ∗server, int port, int type)

  Exception function for **linemonitor_server**() (p. 31)*, print exception code and meaning on the screen and initiate appropriate actions if necessary.*

- **thread_wait_for_linemonitor** ()

  Start the **linemonitor_server**() (p. 31) *– waiting for a connection in a background-thread.*

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

  test0001sockts: see test0001sockts.c.

Variables

- GtkWidget ∗ **gs_application**
- int **robot_x** = 50
- int **robot_y** = 0
- **ROBOT_POSITION robot_position**
- int **serverport**

- int **sock**
- int **soft_msec**
- int **hard_msec**
- int **wait_msec**
- int **ems_fd**

4.6.1 Detailed Description

Server/User-Side Application with GUI to control the robot.

4.6.2 Function Documentation

4.6.2.1 void gs_button_press_event (GtkWidget ∗ *widget*, gpointer *data*)

This function is called if a button is pressed.

Parameters:

widget (GtkWidget ∗) pointer to affected object (button).

event (GdkEventMotion ∗) pointer to structure which describes the event.

4.6.2.2 void gs_da_expose_event (GtkWidget ∗ *widget*, gpointer *data*)

This function is called by gtk if the drawing area (da) needs to be redrawn.

For example if it becomes visible.

Parameters:

widget (GtkWidget ∗) pointer to affected object (in this case the drawing area).

data (gpointer) pointer to additional data from gtk.

4.6.2.3 void gs_delete_event (GtkWidget ∗ *widget*, gpointer *data*)

This function will be called be gtk when the window gets a closing command, it makes sure that the interface is down and the program is finished.

Parameters:

widget (GtkWidget ∗) pointer to affected object (for example: window).

data (gpointer) pointer to additional data from gtk.

4.6.2.4 linemonitor_exception (char ∗ *server*, int *port*, int *type*)

Exception function for **linemonitor_server**() (p. 31), print exception code and meaning on the screen and initiate appropriate actions if necessary.

See **linemonitor_server**() (p. 31) API.

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

test0001sockts: see test0001sockts.c.

4.6.2.6 thread_wait_for_linemonitor ()

Start the **linemonitor_server**() (p. 31) – waiting for a connection in a background-thread.

This has to be a thread because otherwise the system would block. It would wait for a connection while it is supposed to connect itself.

4.6.3 Variable Documentation

4.6.3.1 int ems_fd

4.6.3.2 GtkWidget* gs_application

4.6.3.3 int hard_msec

4.6.3.4 struct ROBOT_POSITION robot_position

4.6.3.5 int robot_x = 50

4.6.3.6 int robot_y = 0

4.6.3.7 int serverport

4.6.3.8 int sock

4.6.3.9 int soft_msec

4.6.3.10 int wait_msec

4.7 interface.c File Reference

Implementation of an example interface to a simple robot with two independent axies.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <asm/io.h>
#include <time.h>
#include <sys/wait.h>
```

Defines

- #define **interface_ymax_time** 20000
  Time it rakes to drive to robot form y_min to y_max.
- #define **interface_xmax_time** 20000
  Time it rakes to drive to robot form x_min to x_max.
- #define **interface_ymax** 100
  Maximal Y value (cosidered as 100 percent).
- #define **interface_xmax** 100
  Maximal X value (cosidered as 100 percent).
- #define **interface_ioport** 0x378
  IO Port to which lowest nible to robot is connected.

Functions

- unsigned char **input** (int addr)
  Read a byte from an IO port.
- unsigned char **output** (int addr, unsigned char out)
  Write a byte to an IO port.
- void **msleep** (int msec)
  Wait a specified number of milliseconds.
- int **getmax** (int a, int b)
  Get to highest number out of two input numbers.
- int **getmin** (int a, int b)
  Get to lowest number out of two input numbers.
- void **interface_drive** (int h, int v)
  Drive the robot in a specified direction.

- void **interface_init** (int mode)
  Initialze the interface and drive the robot to the start position.
- void **interface_driveto** (int x, int y)
  Drive to robot to absolute coordinates.
- void **interface_stop** ()
  Stop the interface, switch all off.

Variables

- int **interface_x**
  current X possition of robot (global).
- int **interface_y**
  current X possition of robot (global).
- int **interface_mode**
  mode of interface: 0: normal, 1: simulation (do all except to drive the robot), 2: simulation with position-output 3: Blocked: Do nothing.

4.7.1 Detailed Description

Implementation of an example interface to a simple robot with two independent axies.

The robot has four inputs whish are connected to the lower nible on IO port "interface_ioport". The bits are connected in this way (the signals a high-active):

Bit 0: Drive Up wires: switch to GND: yellow-green; +24V: gray-black

Bit 1: Dirve down wires: switch to GND: red-green; +24V: orange-black

Bit 2: Dirve right wires: switch to GND: green-red; +24V: yellow-black

Bit 3: Dirve left wires: switch to GND: white-red; +24V: red-blue

power wires: GND: blue; +24V: red

This interface assumes a linear dependency betwenn the coverence of distance and driving time. The 0,0 coordinates is left,bottom.

User functions are:

interface_init() (p. 24) - Initialze the interface and drive the robot to the start position (X=undefined; Y=0).

interface_driveto(int x, int y) (p. 24) - Drive the robot to the absolute coordinates x,y.

4.7.2 Define Documentation

4.7.2.1 #define interface_ioport 0x378

IO Port to which lowest nible to robot is connected.

4.7.2.2 #define interface_xmax 100

Maximal X value (cosidered as 100 percent).

4.7.2.3 #define interface_xmax_time 20000

Time it rakes to drive to robot form x_min to x_max.

4.7.2.4 #define interface_ymax 100

Maximal Y value (cosidered as 100 percent).

4.7.2.5 #define interface_ymax_time 20000

Time it rakes to drive to robot form y_min to y_max.

4.7.3 Function Documentation

4.7.3.1 int getmax (int *a*, int *b*)

Get to highest number out of two input numbers.

Parameters:
- ***a*** (int) first input number
- ***b*** (int) second input number

Returns:
(int) the highest of the input numbers

4.7.3.2 int getmin (int *a*, int *b*)

Get to lowest number out of two input numbers.

Parameters:
- ***a*** (int) first input number
- ***b*** (int) second input number

Returns:
(int) the lowest of the input numbers

4.7.3.3 unsigned char input (int *addr*)

Read a byte from an IO port.

Parameters:
- ***addr*** (int): address of port to read

Returns:
(unsigned char) read byte

4.7.3.4 void interface_drive (int *h*, int *v*)

Drive the robot in a specified direction.

Any axies can be zero, greater than zero or less than zero, in this cases the robot will not driven, driven to increase to position (up[v,y] or right[h,x]) and decrease the position (down[-v,-y] or left[-h,-x]).

Parameters:

h (int) horizontal or X axias.

v (int) vertical or Y axias.

4.7.3.5 void interface_driveto (int *x*, int *y*)

Drive to robot to absolute coordinates.

Parameters:

x (int): absolute x coordinate

y (int): absolute y coordinate

4.7.3.6 void interface_init (int *mode*)

Initialze the interface and drive the robot to the start position.

4.7.3.7 void interface_stop ()

Stop the interface, switch all off.

4.7.3.8 void msleep (int *msec*)

Wait a specified number of milliseconds.

Parameters:

msec (int): milliseconds to wait

4.7.3.9 unsigned char output (int *addr*, unsigned char *out*)

Write a byte to an IO port.

Parameters:

addr (int): address of port to write onto

out (unsigned char): byte to write

Returns:

(unsigned char) written byte

4.7.4 Variable Documentation

4.7.4.1 int interface_mode

mode of interface: 0: normal, 1: simulation (do all except to drive the robot), 2: simulation with position-output 3: Blocked: Do nothing.

4.7.4.2 int interface_x

current X possition of robot (global).

4.7.4.3 int interface_y

current X possition of robot (global).

4.8 libcomm.c File Reference

Main part of libcomm.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/poll.h>
#include <string.h>
#include "libcomm.h"
#include "md5.h"
#include <pthread.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
```

Functions

- void **socket_accept_thread** (struct **LIBCOMMPTHREADP** *libcommpthreadp)

  This is a part of **socket_accept**() (p. 32) *and must not called from the user.*

- int **socket_accept** (int sockport, int id, void(*socket_accept_do)(int fd, int id, char *pip, struct sockaddr_in their_addr))

  Start a new thread, wait for connections and start socket_accept_do() when someone connects.

- int **socket_bind** (int port, int cqueue)

  Bind a socket to a port (Server side).

- int **socket_connect** (char *host, int port)

  Connect a TCP-stream to a server (Client side).

- char * **block_random** (char *buf, int size)

  Get random numbers/bytes.

- void **thread1** (struct **LIBCOMMPTHREADS** *libcommpthreads)

  This is a part of **block_call**() (p. 28) *and must not called from the user.*

- int **block_call** (int fd, int id, int term, void(*block_call_do)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term), void(*block_call_term)(int fd, int id))

  Start in thread, wait for a datablock and call the function block_call_do() if one was received or block_call_term() when the connection terminates.

- int **block_ifdata** (int fd)

This function tests if new data is available to read on a stream.

- char * **block_receive_poll** (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

  Test if is there data available on the socket and read it in a buffer if there is.

- char * **block_receive** (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

  Receive a block (composition of: type, size of datablock and datablock) from a socket.

- int **block_receive_integer** (int fd, unsigned int *recvi)

  Receive an integer (two bytes; 16Bit) from the socket.

- int **block_receive_nbytes** (int fd, char *buf, int n)

  Receive n bytes from socket.

- int **block_send** (int fd, unsigned int type, char *buf, unsigned int size)

  Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

- void **free_authinfo** (struct **AUTHINFO** *destroy)

  Free the memory space which is used by an **AUTHINFO** (p. 5) *structure.*

- int **socket_md5auth** (int fd, char *netname, char *name, struct **AUTHINFO** **plocallogin, struct **AUTHINFO** **premotelogin)

  Do both side authentification.

- **AUTHINFO** * **getauthinfo** (char *netname, char *name)

  Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

- void **linemonitor_server_thread** (struct **LINEMONITOR_THREAD_DATA** *linemonitor_thread_data)

  Thread used by **linemonitor_server**() (p. 31) *NOT for direct usage.*

- int **linemonitor_server** (int port, int **soft_msec**, int **hard_msec**, int **wait_msec**, void(*linemonitor_exception)(char *server, int port, int type))

  Monitor if the "line" is fast enough: Server Application.

- void **linemonitor_emergencystop** (int **sock**)

  Sends an "Emergency Stop" to the client's side, **linemonitor**() (p. 31) *will produce an "Emergency Stop" exception (type 4).*

- int **linemonitor_thread** (struct **LINEMONITOR_THREAD_DATA** *linemonitor_thread_data)

  Thread used by **linemonitor**() (p. 31) *NOT for direct usage.*

- int **linemonitor** (char *server, int port, int **soft_msec**, int **hard_msec**, int **wait_msec**, void(*linemonitor_exception)(char *server, int port, int type))

  Monitor if the "line" is fast enough: Client/Robot Application.

4.8.1 Detailed Description

Main part of libcomm.

4.8.2 Function Documentation

4.8.2.1 int block_call (int *fd*, int *id*, int *term*, void(∗ *block_call_do*)(int fd, int id, unsigned int type, char ∗buf, unsigned int size, int term), void(∗ *block_call_term*)(int fd, int id))

Start in thread, wait for a datablock and call the function block_call_do() if one was received or block_call_term() when the connection terminates.

Parameters:

fd (int) descriptor of socket

id (int) arbitrary id of background process / thread

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

block_call_do (int fd, int id, unsigned int type, char ∗buf, unsigned int size, int term) (function) this function is called if a datablock was received. fd, id and term are the same as in **block_call**() (p. 28). type describes the type of the received datablock, buf is a pointer to this datablock and size is the number of bytes of the datablock

block_call_term (int fd, int id) (function) this function is called if the connection terminates. fd and id are the same as in **block_call**() (p. 28).

Returns:

If all right zero otherwise non zero.

4.8.2.2 int block_ifdata (int *fd*)

This function tests if new data is available to read on a stream.

Parameters:

fd (int) discriptor of stream to test

Returns:

(int) 1: Data to read; 0: No data to read

4.8.2.3 char∗ block_random (char ∗ *buf*, int *size*)

Get random numbers/bytes.

This function read random numbers/bytes from /dev/urandom and stort this bytes in a buffer.

Parameters:

buf (char ∗) in which the bytes will be stored. If this parameter is equal to NULL dynamic memory will be allocated.

size an integer, specifies ths size of the buffer (the number of the random bytes). WARNING: If buf is not equal to null, n∗(size) bytes will be stored in this buffer without any check of ths size of this buf.

Returns:

(char ∗) a pointer to the buffer in which the random bytes are stored.

4.8.2.4 char* block_receive (int *fd*, unsigned int * *type*, char * *buf*, unsigned int * *size*, unsigned int *maxsize*, int *term*)

Receive a block (composition of: type, size of datablock and datablock) from a socket.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int *) pointer to integer, this value can be used as buyer's option

buf (char *) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int *) pointer to integer in which the size of the received datablock is saved.

maxsize (unsigned int *) describes size of buf. This parameter will be ignored if bus is equal to null.

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:

(char *) pointer to buffer which contains the received datablock; NULL if fail.

4.8.2.5 int block_receive_integer (int *fd*, unsigned int * *recvi*)

Receive an integer (two bytes; 16Bit) from the socket.

Parameters:

fd (int) descriptor of socket

recvi (unsigned int *) pointer to integer in which the received integer is saved.

Returns:

(int) 2: OK; -1: fail

4.8.2.6 int block_receive_nbytes (int *fd*, char * *buf*, int *n*)

Receive n bytes from socket.

Parameters:

fd (integer) descriptor of socket

buf (char *) buffer for saving the received bytes

n (integer) number of bytes to receive

Returns:

(integer) n: OK; -1 fial

4.8.2.7 char* block_receive_poll (int *fd*, unsigned int * *type*, char * *buf*, unsigned int * *size*, unsigned int *maxsize*, int *term*)

Test if is there data available on the socket and read it in a buffer if there is.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int ∗) pointer to integer, this value can be used as buyer's option

buf (char ∗) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int ∗) pointer to integer in which the size of the received datablock is saved.

maxsize (unsigned int ∗) describes size of buf. This parameter will be ignored if bus is equal to null.

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:

(char ∗) pointer to buffer which contains the received datablock; NULL if fail; 1 if no data available.

4.8.2.8 int block_send (int *fd*, unsigned int *type*, char ∗ *buf*, unsigned int *size*)

Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of the socket to which buf should send

type (unsigned int) This value can be used as buyer's option

buf (char ∗) which should be send

Returns:

number of sent bytes, -1 if an error is occurt.

4.8.2.9 void free_authinfo (struct AUTHINFO ∗ *destroy*)

Free the memory space which is used by an **AUTHINFO** (p. 5) structure.

Parameters:

struct **AUTHINFO** (p. 5) ∗) pointer to structure to destroy.

4.8.2.10 struct AUTHINFO∗ getauthinfo (char ∗ *netname*, char ∗ *name*)

Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

Parameters:

netname (char ∗) specify the network name (may IP). NULL not specified.

name (char ∗) specity the login name. NULL not specified.

Returns:

(struct **AUTHINFO** (p. 5) ∗) the first entry from authfile which matches network name OR login name. If both values are NULL, the first entry of the authfile is given back.

4.8.2.11 int linemonitor (char ∗ *server*, int *port*, int *soft_msec*, int *hard_msec*, int *wait_msec*, void(∗ *linemonitor_exception*)(char ∗server, int port, int type))

Monitor if the "line" is fast enough: Client/Robot Application.

This function opens a socket stream, sents pings/bytes and wait for them to come back. The soft-timeout will called after soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted. wait_msec specifies the time which is waited after a ping is received befor the next one will be launched.

Parameters:

server (char ∗) server to be connected

port (int) port to be connected

soft_msec (int) timeout in milliseconds which causes soft-real-time exception.

hard_msec (int) timeout in milliseconds which causes hard-real-time exception.

wait_msec (int) timeout for resent – sending of the next ping.

linemonitor_exception (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char ∗) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception, 3: Transmission Fault, 4: Emergency Stop.

4.8.2.12 void linemonitor_emergencystop (int *sock*)

Sends an "Emergency Stop" to the client's side, **linemonitor**() (p. 31) will produce an "Emergency Stop" exception (type 4).

4.8.2.13 int linemonitor_server (int *port*, int *soft_msec*, int *hard_msec*, int *wait_msec*, void(∗ *linemonitor_exception*)(char ∗server, int port, int type))

Monitor if the "line" is fast enough: Server Application.

This function opens a port and wait for the first connection on this port. All data/pings which is sent bei this first connection will be sent back. The soft-timeout will called after wait_msec AND soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted.

Parameters:

port (int) port which should be listend

soft_msec (int) timeout in milliseconds which causes soft-real-time exception.

hard_msec (int) timeout in milliseconds which causes hard-real-time exception.

wait_msec (int) timeout for resent – sending of the next ping.

linemonitor_exception (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char ∗) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception.

Returns:

(int) Filediscriptor to the used socket. Only for usage with **linemonitor_emergencystop**() (p. 31).

4.8.2.14 void linemonitor_server_thread (struct LINEMONITOR_THREAD_DATA ∗ *linemonitor_thread_data*)

Thread used by **linemonitor_server**() (p. 31) NOT for direct usage.

4.8.2.15 int linemonitor_thread (struct LINEMONITOR_THREAD_DATA ∗ *linemonitor_thread_data*)

Thread used by **linemonitor**() (p. 31) NOT for direct usage.

4.8.2.16 int socket_accept (int *sockport*, int *id*, void(∗ *socket_accept_do*)(int fd, int id, char ∗pip, struct sockaddr_in their_addr))

Start a new thread, wait for connections and start socket_accept_do() when someone connects.

Parameters:

sockport (int) descriptor of a tcp socket/port from **socket_bind**() (p. 32)

id (int) arbitrary id of background process / thread. (May be it is a good idea to use the portnumber.)

aocket_accept_do (int fd, int id, char ∗pip, struct sockaddr_in their_addr) (function) this function is called if somebody connects. fd is the descriptor of the new socket to the connected tcp-tream. id is the same as in **socket_accept**() (p. 32). pip contains the ip-address of the connected client. The structure their_addr contails all known information about the connected client.

Returns:

If all right zero otherwise non zero.

4.8.2.17 void socket_accept_thread (struct LIBCOMMPTHREADP ∗ *libcommpthreadp*)

This is a part of **socket_accept**() (p. 32) and must not called from the user.

This function is the thread which is started from **socket_accept**() (p. 32) and runs in background.

4.8.2.18 int socket_bind (int *port*, int *cqueue*)

Bind a socket to a port (Server side).

This function creates an Socket and binds it to a local port.

Parameters:

port an integer which specifies the port

cqueue an integer how many pending connections queue will hold in the waiting queue.

Returns:

The File Descriptor (FD) which allows access to the bound port.

4.8.2.19 int socket_connect (char ∗ *host*, int *port*)

Connect a TCP-stream to a server (Client side).

Creates a socket and connect it over a TCP-stream to the specified port on the specified server.

Parameters:

host a string (char ∗) which specifies the name or the IP-address of the server.

port an integer which specifies the port on the server.

Returns:

The File Descriptor (FD) which allows access to the TCP-stream-socket or -1 if the connection fails.

4.8.2.20 int socket_md5auth (int *fd*, char ∗ *netname*, char ∗ *name*, struct AUTHINFO ∗∗ *plocallogin*, struct AUTHINFO ∗∗ *premotelogin*)

Do both side authentification.

This function is usually called just after a socket stream is established. The function must be called on both sides. Both sides following these steps:

1. get auth info ([login] name, passwd) by using **getauthinfo**() (p. 30) from name or netname for remote login

2. generate random numbers

3. exchange (first send, then receive) login names

4. exchange random numbers

5. calculate md5 checksum over the random numbers (received from other side) and the remote passwd.

6. exchange md5 checksums

7. get auth info from name (received from other side) for local login

8. calculate md5 checksum over the local random numbers and the local passwd.

9. check login – compare the received md5sum (6.) with the generated one (8.); send acknowledgement

10. receive remote acknowledgement

11. return suitable values

Parameters:

fd (int) describes the socket on which the authentication has to be done

netname (char ∗) use netname to resolve [login] name and passwd of the remote machine (NULL: not specified)

netname (char ∗) use [login] name to resolve passwd of the remote machine (NULL: not specified; both NULL use first entry in file, see **getauthinfo**() (p. 30))

plocallogin (struct **AUTHINFO** (p. 5) ∗∗) (pointer to pointer to an **AUTHINFO** (p. 5) struct) in this (double pointed) struct the local authinfo will be loaded, if the parameter is not null.

premotelogin (struct **AUTHINFO** (p. 5) ∗∗) in this (double pointed) struct the remote authinfo will be loaded, if the parameter is not null.

Returns:

(int) 0: Authentication/Login OK; -1: remote login error; -2: login error on both sides; -3: local login error; -4: other (network) error; -5: cannot load remote auth info; -6: cannot load local auth info;

4.8.2.21 void thread1 (struct LIBCOMMPTHREADS ∗ *libcommpthreads*)

This is a part of **block_call**() (p. 28) and must not called from the user.

This function is the thread which is started from **block_call**() (p. 28) and runs in background.

Parameters:

libcommpthreads (struct **LIBCOMMPTHREADS** (p. 7) ∗) holds pointers to the functions to be call, fd (socket discriptor) and id.

4.9 libcomm.h File Reference

Definitions for libcomm.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
```

Data Structures

- struct **AUTHINFO**
- struct **LIBCOMMPTHREADP**
- struct **LIBCOMMPTHREADS**
- struct **LINEMONITOR_THREAD_DATA**

Defines

- #define **true** 1
- #define **false** 0
- #define **authfile0** "./libcomm_md5auth.pwd"
- #define **authfile1** "/etc/libcomm_md5auth.pwd"
- #define **authfilefieldseperator** ':'
- #define **authfilemaxlinelenght** 4096
- #define **authrandomstringsize** (int) 16
- #define **authmessagetype** 65535

Functions

- int **socket_md5auth** (int fd, char *netname, char *name, struct **AUTHINFO** **locallogin, struct **AUTHINFO** **remotelogin)

  Do both side authentification.

- **AUTHINFO** * **getauthinfo** (char *netname, char *name)

  Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

- void **socket_accept_thread** (struct **LIBCOMMPTHREADP** *libcommpthreadp)

  This is a part of **socket_accept**() (p. 32) *and must not called from the user.*

- int **socket_accept** (int sockport, int id, void(*socket_accept_do)(int fd, int id, char *pip, struct sockaddr_in their_addr))

  Start a new thread, wait for connections and start socket_accept_do() when someone connects.

- void **thread1** (struct **LIBCOMMPTHREADS** *libcommpthreads)

  This is a part of **block_call**() (p. 28) *and must not called from the user.*

- int **block_call** (int fd, int id, int term, void(*block_call_do)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term), void(*block_call_term)(int fd, int id))

  Start in thread, wait for a datablock and call the function block_call_do() if one was received or block_call_term() when the connection terminates.

- char * **block_receive_poll** (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

  Test if is there data available on the socket and read it in a buffer if there is.

- char * **block_receive** (int fd, unsigned int *type, char *buf, unsigned int *size, unsigned int maxsize, int term)

  Receive a block (composition of: type, size of datablock and datablock) from a socket.

- int **block_receive_integer** (int fd, unsigned int *recvi)

  Receive an integer (two bytes; 16Bit) from the socket.

- int **block_receive_nbytes** (int fd, char *buf, int n)

  Receive n bytes from socket.

- int **block_send** (int fd, unsigned int type, char *buf, unsigned int size)

  Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

- char * **block_random** (char *buf, int size)

  Get random numbers/bytes.

- int **socket_bind** (int port, int cqueue)

  Bind a socket to a port (Server side).

- int **socket_connect** (char *host, int port)

  Connect a TCP-stream to a server (Client side).

- void **linemonitor_server_thread** (struct **LINEMONITOR_THREAD_DATA** *linemonitor_thread_data)

  Thread used by **linemonitor_server***()* (p. 31) *NOT for direct usage.*

- int **linemonitor_server** (int port, int **soft_msec**, int **hard_msec**, int **wait_msec**, void(*linemonitor_exception)(char *server, int port, int type))

  Monitor if the "line" is fast enough: Server Application.

- void **linemonitor_emergencystop** (int **sock**)

  Sends an "Emergency Stop" to the client's side, **linemonitor***()* (p. 31) *will produce an "Emergency Stop" exception (type 4).*

- int **linemonitor_thread** (struct **LINEMONITOR_THREAD_DATA** *linemonitor_thread_data)

  Thread used by **linemonitor***()* (p. 31) *NOT for direct usage.*

- int **linemonitor** (char *server, int port, int **soft_msec**, int **hard_msec**, int **wait_msec**, void(*linemonitor_exception)(char *server, int port, int type))

  Monitor if the "line" is fast enough: Client/Robot Application.

4.9.1 Detailed Description

Definitions for libcomm.

4.9.2 Define Documentation

4.9.2.1 #define authfile0 "./libcomm_md5auth.pwd"

4.9.2.2 #define authfile1 "/etc/libcomm_md5auth.pwd"

4.9.2.3 #define authfilefieldseperator ':'

4.9.2.4 #define authfilemaxlinelenght 4096

4.9.2.5 #define authmessagetype 65535

4.9.2.6 #define authrandomstringsize (int) 16

4.9.2.7 #define false 0

4.9.2.8 #define true 1

4.9.3 Function Documentation

4.9.3.1 int block_call (int *fd*, int *id*, int *term*, void(* *block_call_do*)(int fd, int id, unsigned int type, char *buf, unsigned int size, int term), void(* *block_call_term*)(int fd, int id))

Start in thread, wait for a datablock and call the function block_call_do() if one was received or block_call_term() when the connection terminates.

Parameters:

fd (int) descriptor of socket

id (int) arbitrary id of background process / thread

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

block_call_do (int fd, int id, unsigned int type, char *buf, unsigned int size, int term) (function) this function is called if a datablock was received. fd, id and term are the same as in **block_call**() (p. 28). type describes the type of the received datablock, buf is a pointer to this datablock and size is the number of bytes of the datablock

block_call_term (int fd, int id) (function) this function is called if the connection terminates. fd and id are the same as in **block_call**() (p. 28).

Returns:

If all right zero otherwise non zero.

4.9.3.2 char* block_random (char * *buf*, int *size*)

Get random numbers/bytes.

This function read random numbers/bytes from /dev/urandom and stort this bytes in a buffer.

Parameters:

buf (char ∗) in which the bytes will be stored. If this parameter is equal to NULL dynamic memory will be allocated.

size an integer, specifies ths size of the buffer (the number of the random bytes). WARNING: If buf is not equal to null, n∗(size) bytes will be stored in this buffer without any check of ths size of this buf.

Returns:

(char ∗) a pointer to the buffer in which the random bytes are stored.

4.9.3.3 char∗ block_receive (int *fd*, unsigned int ∗ *type*, char ∗ *buf*, unsigned int ∗ *size*, unsigned int *maxsize*, int *term*)

Receive a block (composition of: type, size of datablock and datablock) from a socket.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int ∗) pointer to integer, this value can be used as buyer's option

buf (char ∗) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int ∗) pointer to integer in which the size of the received datablock is saved.

maxsize (unsigned int ∗) describes size of buf. This parameter will be ignored if bus is equal to null.

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:

(char ∗) pointer to buffer which contains the received datablock; NULL if fail.

4.9.3.4 int block_receive_integer (int *fd*, unsigned int ∗ *recvi*)

Receive an integer (two bytes; 16Bit) from the socket.

Parameters:

fd (int) descriptor of socket

recvi (unsigned int ∗) pointer to integer in which the received integer is saved.

Returns:

(int) 2: OK; -1: fail

4.9.3.5 int block_receive_nbytes (int *fd*, char ∗ *buf*, int *n*)

Receive n bytes from socket.

Parameters:

fd (integer) descriptor of socket

buf (char ∗) buffer for saving the received bytes

n (integer) number of bytes to receive

Returns:

(integer) n: OK; -1 fial

4.9.3.6 char∗ block_receive_poll (int *fd*, unsigned int ∗ *type*, char ∗ *buf*, unsigned int ∗ *size*, unsigned int *maxsize*, int *term*)

Test if is there data available on the socket and read it in a buffer if there is.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of socket

type (unsigned int ∗) pointer to integer, this value can be used as buyer's option

buf (char ∗) buffer for datablock. Memory will be allocated if this parameter is equal to null.

size (unsigned int ∗) pointer to integer in which the size of the received datablock is saved.

maxsize (unsigned int ∗) describes size of buf. This parameter will be ignored if bus is equal to null.

term (int) 0: do not terminate the buffer, 1: terminate the buffer by appending a 0x00.

Returns:

(char ∗) pointer to buffer which contains the received datablock; NULL if fail; 1 if no data available.

4.9.3.7 int block_send (int *fd*, unsigned int *type*, char ∗ *buf*, unsigned int *size*)

Send a block (composition of: type, size of datablock and datablock (buf)) to a socket.

WARNING: The integers (type and size; excluding fd) are only 16 bit values (0 - 65535).

Parameters:

fd (int) descriptor of the socket to which buf should send

type (unsigned int) This value can be used as buyer's option

buf (char ∗) which should be send

Returns:

number of sent bytes, -1 if an error is occurt.

4.9.3.8 struct AUTHINFO∗ getauthinfo (char ∗ *netname*, char ∗ *name*)

Load authentication informations (netname, name, passwd, keyencrypt, keydecrypt) from authfile.

Parameters:

netname (char ∗) specify the network name (may IP). NULL not specified.

name (char ∗) specity the login name. NULL not specified.

Returns:

(struct **AUTHINFO** (p. 5) ∗) the first entry from authfile which matches network name OR login name. If both values are NULL, the first entry of the authfile is given back.

4.9.3.9 int linemonitor (char * *server*, int *port*, int *soft_msec*, int *hard_msec*, int *wait_msec*, void(* *linemonitor_exception*)(char *server, int port, int type))

Monitor if the "line" is fast enough: Client/Robot Application.

This function opens a socket stream, sents pings/bytes and wait for them to come back. The soft-timeout will called after soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted. wait_msec specifies the time which is waited after a ping is received befor the next one will be launched.

Parameters:

server (char *) server to be connected

port (int) port to be connected

soft_msec (int) timeout in milliseconds which causes soft-real-time exception.

hard_msec (int) timeout in milliseconds which causes hard-real-time exception.

wait_msec (int) timeout for resent – sending of the next ping.

linemonitor_exception (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char *) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception, 3: Transmission Fault, 4: Emergency Stop.

4.9.3.10 void linemonitor_emergencystop (int *sock*)

Sends an "Emergency Stop" to the client's side, **linemonitor**() (p. 31) will produce an "Emergency Stop" exception (type 4).

4.9.3.11 int linemonitor_server (int *port*, int *soft_msec*, int *hard_msec*, int *wait_msec*, void(* *linemonitor_exception*)(char *server, int port, int type))

Monitor if the "line" is fast enough: Server Application.

This function opens a port and wait for the first connection on this port. All data/pings which is sent bei this first connection will be sent back. The soft-timeout will called after wait_msec AND soft_msec is timeouted. The hard-timeout will called after soft-timeout was called AND hard_msec is timeouted.

Parameters:

port (int) port which should be listend

soft_msec (int) timeout in milliseconds which causes soft-real-time exception.

hard_msec (int) timeout in milliseconds which causes hard-real-time exception.

wait_msec (int) timeout for resent – sending of the next ping.

linemonitor_exception (pointer to function) This function will be called if an exception occurs. It becomes the following parameters: server name (char *) which is always null, port (int): listend port and type (int) of exception which can be: 0: Connicion Fault, 1: Soft Real Time Exception, 2: HARD Real Time Exception.

Returns:

(int) Filediscriptor to the used socket. Only for usage with **linemonitor_emergencystop**() (p. 31).

4.9.3.12 void linemonitor_server_thread (struct LINEMONITOR_THREAD_DATA ∗ *linemonitor_thread_data*)

Thread used by **linemonitor_server**() (p. 31) NOT for direct usage.

4.9.3.13 int linemonitor_thread (struct LINEMONITOR_THREAD_DATA ∗ *linemonitor_thread_data*)

Thread used by **linemonitor**() (p. 31) NOT for direct usage.

4.9.3.14 int socket_accept (int *sockport*, int *id*, void(∗ *socket_accept_do*)(int fd, int id, char ∗pip, struct sockaddr_in their_addr))

Start a new thread, wait for connections and start socket_accept_do() when someone connects.

Parameters:

sockport (int) descriptor of a tcp socket/port from **socket_bind**() (p. 32)

id (int) arbitrary id of background process / thread. (May be it is a good idea to use the portnumber.)

aocket_accept_do (int fd, int id, char ∗pip, struct sockaddr_in their_addr) (function) this function is called if somebody connects. fd is the descriptor of the new socket to the connected tcp-tream. id is the same as in **socket_accept**() (p. 32). pip contains the ip-address of the connected client. The structure their_addr contails all known information about the connected client.

Returns:

If all right zero otherwise non zero.

4.9.3.15 void socket_accept_thread (struct LIBCOMMPTHREADP ∗ *libcommpthreadp*)

This is a part of **socket_accept**() (p. 32) and must not called from the user.

This function is the thread which is started from **socket_accept**() (p. 32) and runs in background.

4.9.3.16 int socket_bind (int *port*, int *cqueue*)

Bind a socket to a port (Server side).

This function creates an Socket and binds it to a local port.

Parameters:

port an integer which specifies the port

cqueue an integer how many pending connections queue will hold in the waiting queue.

Returns:

The File Descriptor (FD) which allows access to the bound port.

4.9.3.17 int socket_connect (char * *host*, int *port*)

Connect a TCP-stream to a server (Client side).

Creates a socket and connect it over a TCP-stream to the specified port on the specified server.

Parameters:

host a string (char *) which specifies the name or the IP-address of the server.

port an integer which specifies the port on the server.

Returns:

The File Descriptor (FD) which allows access to the TCP-stream-socket or -1 if the connection fails.

4.9.3.18 int socket_md5auth (int *fd*, char * *netname*, char * *name*, struct AUTHINFO ** *plocallogin*, struct AUTHINFO ** *premotelogin*)

Do both side authentification.

This function is usually called just after a socket stream is established. The function must be called on both sides. Both sides following these steps:

1. get auth info ([login] name, passwd) by using **getauthinfo**() (p. 30) from name or netname for remote login

2. generate random numbers

3. exchange (first send, then receive) login names

4. exchange random numbers

5. calculate md5 checksum over the random numbers (received from other side) and the remote passwd.

6. exchange md5 checksums

7. get auth info from name (received from other side) for local login

8. calculate md5 checksum over the local random numbers and the local passwd.

9. check login – compare the received md5sum (6.) with the generated one (8.); send acknowledgement

10. receive remote acknowledgement

11. return suitable values

Parameters:

fd (int) describes the socket on which the authentication has to be done

netname (char *) use netname to resolve [login] name and passwd of the remote machine (NULL: not specified)

netname (char *) use [login] name to resolve passwd of the remote machine (NULL: not specified; both NULL use first entry in file, see **getauthinfo**() (p. 30))

plocallogin (struct **AUTHINFO** (p. 5) **) (pointer to pointer to an **AUTHINFO** (p. 5) struct) in this (double pointed) struct the local authinfo will be loaded, if the parameter is not null.

premotelogin (struct **AUTHINFO** (p. 5) **) in this (double pointed) struct the remote authinfo will be loaded, if the parameter is not null.

Returns:

(int) 0: Authentication/Login OK; -1: remote login error; -2: login error on both sides; -3: local login error; -4: other (network) error; -5: cannot load remote auth info; -6: cannot load local auth info;

4.9.3.19 void thread1 (struct LIBCOMMPTHREADS * *libcommpthreads*)

This is a part of **block_call**() (p. 28) and must not called from the user.

This function is the thread which is started from **block_call**() (p. 28) and runs in background.

Parameters:

libcommpthreads (struct **LIBCOMMPTHREADS** (p. 7) *) holds pointers to the functions to be call, fd (socket discriptor) and id.

4.10 md5.c File Reference

Implementation of the MD5 algorithm by Ron Rivest, code Colin Plumb (public domain).

```
#include <string.h>
#include "md5.h"
```

Defines

- #define **byteReverse**(buf, len)
- #define **F1**(x, y, z) (z ^ (x & (y ^ z)))
- #define **F2**(x, y, z) F1(z, x, y)
- #define **F3**(x, y, z) (x ^ y ^ z)
- #define **F4**(x, y, z) (y ^ (x | ∼z))
- #define **MD5STEP**(f, w, x, y, z, data, s) (w += f(x, y, z) + data, w = w<<s | w>>(32-s), w += x)

Functions

- void **MD5Init** (struct **MD5Context** ∗ctx)
- void **MD5Update** (struct **MD5Context** ∗ctx, unsigned char const ∗buf, unsigned len)
- void **MD5Final** (unsigned char digest[16], struct **MD5Context** ∗ctx)
- void **MD5Transform** (**uint32** buf[4], **uint32** const in[16])

4.10.1 Detailed Description

Implementation of the MD5 algorithm by Ron Rivest, code Colin Plumb (public domain).

4.10.2 Define Documentation

4.10.2.1 #define byteReverse(buf, len)

4.10.2.2 #define F1(x, y, z) (z ^ (x & (y ^ z)))

4.10.2.3 #define F2(x, y, z) F1(z, x, y)

4.10.2.4 #define F3(x, y, z) (x ^ y ^ z)

4.10.2.5 #define F4(x, y, z) (y ^ (x | ~z))

4.10.2.6 #define MD5STEP(f, w, x, y, z, data, s) (w += f(x, y, z) + data, w = w<<s | w>>(32-s), w += x)

4.10.3 Function Documentation

4.10.3.1 void MD5Final (unsigned char *digest*[16], struct MD5Context * *ctx*)

4.10.3.2 void MD5Init (struct MD5Context * *ctx*)

4.10.3.3 void MD5Transform (uint32 *buf*[4], uint32 const *in*[16])

4.10.3.4 void MD5Update (struct MD5Context * *ctx*, unsigned char const * *buf*, unsigned *len*)

4.11 md5.h File Reference

Definitions for the Implementation of the MD5 algorithm by Ron Rivest, code Colin Plumb (public domain).

Data Structures

- struct **MD5Context**

Typedefs

- typedef unsigned long **uint32**
- typedef **MD5Context MD5_CTX**

Functions

- void **MD5Init** (struct **MD5Context** ∗context)
- void **MD5Update** (struct **MD5Context** ∗context, unsigned char const ∗buf, unsigned len)
- void **MD5Final** (unsigned char digest[16], struct **MD5Context** ∗context)
- void **MD5Transform** (**uint32** buf[4], **uint32** const in[16])

4.11.1 Detailed Description

Definitions for the Implementation of the MD5 algorithm by Ron Rivest, code Colin Plumb (public domain).

4.11.2 Typedef Documentation

4.11.2.1 typedef struct MD5Context MD5_CTX

4.11.2.2 typedef unsigned long uint32

4.11.3 Function Documentation

4.11.3.1 void MD5Final (unsigned char *digest*[16], struct MD5Context ∗ *context*)

4.11.3.2 void MD5Init (struct MD5Context ∗ *context*)

4.11.3.3 void MD5Transform (uint32 *buf*[4], uint32 const *in*[16])

4.11.3.4 void MD5Update (struct MD5Context ∗ *context*, unsigned char const ∗ *buf*, unsigned *len*)

4.12 sleeptest.c File Reference

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

Functions

- **main** ()

4.12.1 Function Documentation

4.12.1.1 main ()

4.13 stoppwatch.c File Reference

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

Functions

- **main** ()

4.13.1 Function Documentation

4.13.1.1 main ()

4.14 tes005realtime.c File Reference

`#include <stdio.h>`

`#include "libcomm.h"`

Functions

- **linemonitor_exception** (char *server, int port, int type)
  Exception Function, print exception code and meaning on the screen.
- **main** (int argc, char *argv[ ])
  test0001sockts: see test0001sockts.c.

4.14.1 Function Documentation

4.14.1.1 linemonitor_exception (char * *server*, int *port*, int *type*)

Exception Function, print exception code and meaning on the screen.

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

test0001sockts: see test0001sockts.c.

4.15 test001interface.c File Reference

Testprogram for the interface to the robot.

`#include <stdio.h>`

Functions

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

4.15.1 Detailed Description

Testprogram for the interface to the robot.

Interface will be initiated, than absolutes coordinates are asked for.

4.15.2 Function Documentation

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

4.16 test001sockets.c File Reference

This program tests the low-level tcp-socket-stream library function.

`#include <stdio.h>`

`#include <arpa/inet.h>`

`#include <sys/types.h>`

`#include <signal.h>`

`#include "../lib/libcomm.h"`

Defines

- #define **PORT** 1234

  Port on which the server is listening and the client trys to connect.

- #define **bufsize** 8192

  How much random bytes should be transfered?

Functions

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

  test0001sockts: see test0001sockts.c.

- **client_test_program** (char *server, int port)

  test0001sockts: see test0001sockts.c, Client Test Programm.

- **server_test_program** (int port)

  test0001sockts: see test0001sockts.c, Server Test Programm.

4.16.1 Detailed Description

This program tests the low-level tcp-socket-stream library function.

4.16.2 Define Documentation

4.16.2.1 #define bufsize 8192

How much random bytes should be transfered?

4.16.2.2 #define PORT 1234

Port on which the server is listening and the client trys to connect.

4.16.3 Function Documentation

4.16.3.1 client_test_program (char * *server*, int *port*)

test0001sockts: see test0001sockts.c, Client Test Programm.

The client test program generates a block of n*(bufsize) random bytes, sents theses bytes to the server, receive a block from server, invert it and compare it with generated block.

BUGS: Receives only one block. If not all data ready and function resv() do noct block, data get lost. Test fails.

Parameters:

server a string which contains the server name

port an integer which specifies the port on the server

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

test0001sockts: see test0001sockts.c.

4.16.3.3 server_test_program (int *port*)

test0001sockts: see test0001sockts.c, Server Test Programm.

The server test program binds a port and waits for connections. If someone connects it reads n*(bufsize) bytes, invert all bits of these bytes and send all back.

BUGS: Receives only one block. If not all data ready and function resv() do noct block, data get lost. Test fails.

Parameters:

port an integer which specifies the port to bind.

4.17 test002guiprototyp.c File Reference

Prototype to explore the programming of gtk (graphical interface).

`#include <time.h>`

`#include <math.h>`

`#include <pthread.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <gtk/gtk.h>`

`#include <glib.h>`

Defines

- #define **CENTER** 150
- #define **LENGTH** 100

Functions

- void **my_callback** (GtkWidget *widget, gpointer data)
- void **drawrobot** (GtkWidget *widget, int x, int y, int ox, int oy)
- void **my_redraw** (GtkWidget *widget, gpointer data)
- void **check_button_press_cb** (GtkWidget *widget, gpointer data)
- int **main** (int argc, char *argv[])

  test0001sockts: see test0001sockts.c.

Variables

- GtkWidget * **spinner_app**

4.17.1 Detailed Description

Prototype to explore the programming of gtk (graphical interface).

Compused of various online-stuff.

4.17.2 Define Documentation

4.17.2.1 #define CENTER 150

4.17.2.2 #define LENGTH 100

4.17.3 Function Documentation

4.17.3.1 void check_button_press_cb (GtkWidget * *widget*, gpointer *data*)

4.17.3.2 void drawrobot (GtkWidget * *widget*, int *x*, int *y*, int *ox*, int *oy*)

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

test0001sockts: see test0001sockts.c.

4.17.3.4 void my_callback (GtkWidget * *widget*, gpointer *data*)

4.17.3.5 void my_redraw (GtkWidget * *widget*, gpointer *data*)

4.17.4 Variable Documentation

4.17.4.1 GtkWidget* spinner_app

4.18 test002integer.c File Reference

This program tests the size and the organisation of an integer on the local machine, with the local compiler.

`#include <stdio.h>`

Functions

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

  test0001sockts: see test0001sockts.c.

4.18.1 Detailed Description

This program tests the size and the organisation of an integer on the local machine, with the local compiler.

4.18.2 Function Documentation

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

test0001sockts: see test0001sockts.c.

4.19 test003block.c File Reference

This program tests the low-level block transfer and authentication functions.

`#include` <`stdio.h`>

`#include` <`arpa/inet.h`>

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

`#include` <`signal.h`>

`#include "libcomm.h"`

`#include` <`unistd.h`>

Defines

- #define **PORT** 1235

  Port on which the server is listening and the client trys to connect.

- #define **bufsize** 8192

  How much random bytes should be transfered?

- #define **threadid** 1234

  The ID which identifies the thread of the **block_call**() (p. 28) *function.*

Functions

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

  test0001sockts: see test0001sockts.c.

- **client_test_program** (char *server, int port, int auth)

  test0001sockts: see test0001sockts.c, Client Test Programm.

- **block_call_do_test** (int fd, int id, unsigned int type, char *buf, unsigned int size, int term)

  testfunction which ist called from **block_call**() (p. 28) *if a message/datablock has received.*

- **block_call_term_test** (int fd, int id)

  testfunction which ist called from **block_call**() (p. 28) *if a the connection terminates.*

- **socket_accept_do_test** (int fd, int id, char *pip, struct sockaddr_in their_addr, int term)

  testfunction which ist called from **socket_accept**() (p. 32) *if someone has connected.*

- **server_test_program** (int port, int mode)

  test0001sockts: see test0001sockts.c, Server Test Programm.

4.19.1 Detailed Description

This program tests the low-level block transfer and authentication functions.

4.19.2 Define Documentation

4.19.2.1 #define bufsize 8192

How much random bytes should be transfered?

4.19.2.2 #define PORT 1235

Port on which the server is listening and the client trys to connect.

4.19.2.3 #define threadid 1234

The ID which identifies the thread of the **block_call**() (p. 28) function.

This value can be coosen arbitrary and is passwd to all called functions.

4.19.3 Function Documentation

4.19.3.1 block_call_do_test (int *fd*, int *id*, unsigned int *type*, char * *buf*, unsigned int *size*, int *term*)

testfunction which ist called from **block_call**() (p. 28) if a message/datablock has received.

4.19.3.2 block_call_term_test (int *fd*, int *id*)

testfunction which ist called from **block_call**() (p. 28) if a the connection terminates.

4.19.3.3 client_test_program (char * *server*, int *port*, int *auth*)

test0001sockts: see test0001sockts.c, Client Test Programm.

The client test program generates a block of n*(bufsize) random bytes, sents theses bytes to the server, receive a block from server, invert it and compare it with generated block.

BUGS: Receives only one block. If not all data ready and function resv() do noct block, data get lost. Test fails.

Parameters:

server (char *) contains the server name

port (int) specifies the port on the server

auth (int) 0: normale test; 1: do authentification before run test.

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

test0001sockts: see test0001sockts.c.

4.19.3.5 server_test_program (int *port*, int *mode*)

test0001sockts: see test0001sockts.c, Server Test Programm.

The server test program binds a port and waits for connections. If someone connects it reads n∗(bufsize) bytes, invert all bits of these bytes and send all back.

BUGS: Receives only one block. If not all data ready and function resv() do noct block, data get lost. Test fails.

Parameters:

port (int) specify the port to bind.

mode (int) select the mode of receiving a message/datablock: 0: blocked mode, use **block_receive**() (p. 29); 1: poll mode, poll with **block_receive_poll**() (p. 29); 2: call function **block_call_do_test**() (p. 57) if a block is received, use **block_call**() (p. 28); 3: call **socket_accept_do_test**() (p. 58) if someone has connected. This function calls **block_call**() (p. 28) which does the same like in 2, use **socket_accept**() (p. 32); 4: testing authenication by using **socket_md5auth**() (p. 33). After this do the same as in 0.

4.19.3.6 socket_accept_do_test (int *fd*, int *id*, char ∗ *pip*, struct sockaddr_in *their_addr*, int *term*)

testfunction which ist called from **socket_accept**() (p. 32) if someone has connected.

4.20 test004md5.c File Reference

`#include <stdio.h>`

`#include "../lib/md5.h"`

Defines

- #define **bufsize** 2048

    Buffersize for reading the file to generate a md5sum of it, for **calculate_md5sum**() (p. 59).

Functions

- char * **calculate_md5sum** (char *filename)

    Calculate the md5sum of a file.

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

Variables

- char **md5sums** [4][16]

4.20.1 Define Documentation

4.20.1.1 #define bufsize 2048

Buffersize for reading the file to generate a md5sum of it, for **calculate_md5sum**() (p. 59).

4.20.2 Function Documentation

4.20.2.1 char* calculate_md5sum (char * *filename*)

Calculate the md5sum of a file.

Parameters:

filename (char *) name of the file of which the md5sum has to be generated

Returns:

(char *) pointer to a static memory area in whish the md5sum ist stored.

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

4.20.3 Variable Documentation

4.20.3.1 char md5sums[4][16]

Initial value:

```
 {
  { 0x9e, 0xb2, 0x86, 0x6e, 0x09, 0xcd, 0x47, 0xed, 0x53, 0x2d, 0x99, 0xb9, 0xc3, 0xcd, 0x4a, 0x86 },
  { 0xd2, 0xb6, 0x5a, 0xe7, 0x29, 0xe7, 0x93, 0x3c, 0x9c, 0xa9, 0x1c, 0x88, 0xdb, 0x7c, 0xd0, 0xfc },
  { 0x4d, 0x3c, 0x9e, 0xff, 0xe9, 0xb4, 0x97, 0x9a, 0x4b, 0xc7, 0xce, 0x2e, 0x45, 0xdf, 0x21, 0xd2 },
  { 0xf5, 0x5c, 0x17, 0x5c, 0x5f, 0x26, 0x98, 0xf9, 0xb2, 0xcc, 0xf7, 0xba, 0x72, 0x01, 0x4b, 0x14 },
}
```