Simulazione Transazioni

Generated by Doxygen 1.9.3

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 block_t Struct Reference

```
#include <common.h>
```

Collaboration diagram for block_t:

Data Fields

- transaction transList [SO_BLOCK_SIZE]
- unsigned int blockIndex
- struct block ∗ next
- struct block ∗ prev

3.1.1 Detailed Description

Definition at line 139 of file common.h.

3.1.2 Field Documentation

3.1.2.1 blockIndex

```
unsigned int blockIndex
```

Definition at line 142 of file common.h.

3.1.2.2 next

```
struct block* next
```

Definition at line 143 of file common.h.

3.1.2.3 prev

```
struct block* prev
```

Definition at line 144 of file common.h.

3.1.2.4 transList

```
transaction transList[SO_BLOCK_SIZE]
```

Definition at line 141 of file common.h.

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

- src/include/common.h

3.2 ledger_t Struct Reference

```
#include <common.h>
```

Collaboration diagram for ledger_t:

Data Fields

- block ∗ head
- unsigned int registryCurrSize

3.2.1 Detailed Description

Definition at line 148 of file common.h.

3.2.2 Field Documentation

3.2.2.1 head

```
block* head
```

Definition at line 150 of file common.h.

3.2.2.2 registryCurrSize

```
unsigned int registryCurrSize
```

Definition at line 151 of file common.h.

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

- src/include/common.h

3.3 message Struct Reference

```
#include <common.h>
```

Collaboration diagram for message:

Data Fields

- long mtype
- transaction userTrans

3.3.1 Detailed Description

Definition at line 132 of file common.h.

3.3.2 Field Documentation

3.3.2.1 mtype

```
long mtype
```

Definition at line 134 of file common.h.

3.3.2.2 userTrans

```
transaction userTrans
```

Definition at line 135 of file common.h.

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

- src/include/common.h

3.4 node_t Struct Reference

```
#include <common.h>
```

Public Types

- enum { available , full }

Data Fields

- pid_t pid
- enum node_t:: { ... } status

3.4.1 Detailed Description

Definition at line 105 of file common.h.

3.4.2 Member Enumeration Documentation

3.4.2.1 anonymous enum

```
anonymous enum
```

Enumerator

available	
full	

Definition at line 108 of file common.h.

3.4.3 Field Documentation

3.4.3.1 pid

```
pid_t pid
```

Definition at line 107 of file common.h.

3.4.3.2

```
enum { ... } status
```

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

- src/include/common.h

3.5 parameters Struct Reference

```
#include <common.h>
```

Data Fields

- unsigned int SO_USER_NUM
- unsigned int SO_NODES_NUM
- unsigned int SO_BUDGET_INIT
- char SO_REWARD
- unsigned long SO_MIN_TRANS_GEN_NSEC
- unsigned long SO_MAX_TRANS_GEN_NSEC
- unsigned int SO_RETRY
- unsigned int SO_TP_SIZE
- unsigned long SO_MIN_TRANS_PROC_NSEC
- unsigned long SO_MAX_TRANS_PROC_NSEC
- unsigned int SO_SIM_SEC
- unsigned int SO_FRIENDS_NUM
- unsigned int SO_HOPS

3.5.1 Detailed Description

Definition at line 77 of file common.h.

3.5.2 Field Documentation

3.5.2.1 SO_BUDGET_INIT

```
unsigned int SO_BUDGET_INIT
```

Definition at line 81 of file common.h.

3.5.2.2 SO_FRIENDS_NUM

```
unsigned int SO_FRIENDS_NUM
```

Definition at line 90 of file common.h.

3.5.2.3 SO_HOPS

```
unsigned int SO_HOPS
```

Definition at line 91 of file common.h.

3.5.2.4 SO_MAX_TRANS_GEN_NSEC

```
unsigned long SO_MAX_TRANS_GEN_NSEC
```

Definition at line 84 of file common.h.

3.5.2.5 SO_MAX_TRANS_PROC_NSEC

```
unsigned long SO_MAX_TRANS_PROC_NSEC
```

Definition at line 88 of file common.h.

3.5.2.6 SO_MIN_TRANS_GEN_NSEC

```
unsigned long SO_MIN_TRANS_GEN_NSEC
```

Definition at line 83 of file common.h.

3.5.2.7 SO_MIN_TRANS_PROC_NSEC

```
unsigned long SO_MIN_TRANS_PROC_NSEC
```

Definition at line 87 of file common.h.

3.5.2.8 SO_NODES_NUM

```
unsigned int SO_NODES_NUM
```

Definition at line 80 of file common.h.

3.5.2.9 SO_RETRY

```
unsigned int SO_RETRY
```

Definition at line 85 of file common.h.

3.5.2.10 SO_REWARD

```
char SO_REWARD
```

Definition at line 82 of file common.h.

3.5.2.11 SO_SIM_SEC

```
unsigned int SO_SIM_SEC
```

Definition at line 89 of file common.h.

3.5.2.12 SO_TP_SIZE

```
unsigned int SO_TP_SIZE
```

Definition at line 86 of file common.h.

3.5.2.13 SO_USER_NUM

```
unsigned int SO_USER_NUM
```

Definition at line 79 of file common.h.

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

- src/include/common.h

3.6 semun Union Reference

```
#include <sem.h>
```

Data Fields

- int val
- struct semid_ds ∗ buf
- unsigned short ∗ array
- struct seminfo ∗ __buf

3.6.1 Detailed Description

Definition at line 31 of file sem.h.

3.6.2 Field Documentation

3.6.2.1 __buf

```
struct seminfo* __buf
```

Definition at line 35 of file sem.h.

3.6.2.2 array

```
unsigned short* array
```

Definition at line 34 of file sem.h.

3.6.2.3 buf

```
struct semid_ds* buf
```

Definition at line 33 of file sem.h.

3.6.2.4 val

```
int val
```

Definition at line 32 of file sem.h.

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

- src/utils/sem.h

3.7 transaction_t Struct Reference

```
#include <common.h>
```

Public Types

- enum { pending , processing , confirmed , aborted }

Data Fields

- struct timespec timestamp
- pid_t sender
- pid_t receiver
- int amount
- int reward
- enum transaction_t:: { ... } status

3.7.1 Detailed Description

Definition at line 116 of file common.h.

3.7.2 Member Enumeration Documentation

3.7.2.1 anonymous enum

```
anonymous enum
```

Enumerator

pending	
processing	
confirmed	
aborted	

Definition at line 123 of file common.h.

3.7.3 Field Documentation

3.7.3.1 amount

```
int amount
```

Definition at line 121 of file common.h.

3.7.3.2 receiver

```
pid_t receiver
```

Definition at line 120 of file common.h.

3.7.3.3 reward

```
int reward
```

Definition at line 122 of file common.h.

3.7.3.4 sender

```
pid_t sender
```

Definition at line 119 of file common.h.

3.7.3.5

```
enum { ...  } status
```

3.7.3.6 timestamp

```
struct timespec timestamp
```

Definition at line 118 of file common.h.

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

- src/include/common.h

3.8 user_t Struct Reference

```
#include <common.h>
```

Public Types

- enum { alive , broke , dead }

Data Fields

- pid_t pid
- enum user_t:: { ... } status

3.8.1 Detailed Description

Definition at line 94 of file common.h.

3.8.2 Member Enumeration Documentation

3.8.2.1 anonymous enum

```
anonymous enum
```

Enumerator

alive	
broke	
dead	

Definition at line 97 of file common.h.

3.8.3 Field Documentation

3.8.3.1 pid

```
pid_t pid
```

Definition at line 96 of file common.h.

3.8.3.2

```
enum { ... } status
```

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

- src/include/common.h

Chapter 4

File Documentation

4.1 src/common.c File Reference

```
#include "include/common.h"
```

Include dependency graph for common.c:

4.2 common.c

Go to the documentation of this file.

```
#include "include/common.h"

ledger *ledger_init()
{
    ledger *newLedger;
    int shmID; /* ID of "ledger" shared memory segment */

    /* -- LEDGER INITIALIZATION --
     * save the ID of our new (IPC_PRIVATE) shared memory segment of size -ledger-
     * smctl will deallocate the shared memory segment only when every process detaches it
     * tells OS that ledger of type ledger is our shared memory of shmID
     */
    shmID = shmget(IPC_PRIVATE, sizeof(newLedger), 0600);
    shmctl(shmID, IPC_RMID, NULL);

    newLedger->head = new_block();
    newLedger->registryCurrSize = 1;

    newLedger = (ledger *)shmat(shmID, NULL, 0);

    return newLedger;
}

block *new_block()
{
    block *newBlock = malloc(sizeof(block));
    transaction reward;
    struct timespec timestamp;

    /* memset(newBlock->transList, 0, SO_BLOCK_SIZE); */
    clock_gettime(CLOCK_REALTIME, &timestamp);

    reward.timestamp = timestamp;
    reward.sender = SELF;
    reward.receiver = getpid();
    reward.amount = 0;
    reward.reward = 0;

    newBlock->transList[0] = reward;
    newBlock->blockIndex = 0;
    newBlock->next = NULL;

    return newBlock;
}

void add_transaction_to_block(block *block, transaction *newTrans, int index)
{
    block->transList[index] = *newTrans; /* ye probably we don’t need a whole ass function for that*/
}
```

4.3 src/include/balance.h File Reference

Functions

- int balance (int budget)

4.3.1 Function Documentation

4.3.1.1 balance()

```
int balance (
             int budget )
```

Definition at line 1 of file balance.h.

4.4 balance.h

Go to the documentation of this file.

```
int balance(int budget)
{
    int unitoCoin_in;
    int unitoCoin_out;
    int unitoCoin_out_notregistered;
    int curr_balance = unitoCoin_in + unitoCoin_out - unitoCoin_out_notregistered;
};

```

4.5 src/include/common.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <sys/types.h>
#include "../utils/debug.h"
#include "../utils/sem.h"
#include "../utils/lists.h"
```

Include dependency graph for common.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct parameters
- struct user_t
- struct node_t
- struct transaction_t
- struct message
- struct block_t
- struct ledger_t

Macros

- #define NULL 0 /∗ thre's a problem with NULL for some reason ∗/
- #define _GNU_SOURCE
- #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
- #define RAND(min, max) ((rand() % (max - min + 1)) + min)
- #define IPC_ERROR -1
- #define SHM_PARAMETERS 1337
- #define SHM_USERS_ARRAY 1338
- #define SHM_NODES_ARRAY 1339
- #define SHM_LEDGER 1340
- #define SEM_MASTER 420
- #define M_QUEUE_KEY 0x5AD
- #define SO_BLOCK_SIZE 100 /∗ number of transaction per block∗/
- #define SO_REGISTRY_SIZE 1000 /∗ max length of consecutive blocks ∗/
- #define SELF -1
- #define EVERYONE_BROKE '$'
- #define USERS_PID_ARGV (atoi(argv[1]))
- #define NODES_PID_ARGV (atoi(argv[2]))
- #define PARAMETERS_ARGV (atoi(argv[3]))
- #define LEDGER_ARGV (atoi(argv[4]))
- #define SEM_ID_ARGV (atoi(argv[5]))
- #define WENT_BROKE 1
- #define MAX_RETRY 2
- #define ERROR -1
- #define SUCCESS 0
- #define TEST_ERROR

Typedefs

- typedef struct user_t user
- typedef struct node_t node
- typedef struct transaction_t transaction
- typedef struct block_t block
- typedef struct ledger_t ledger

Functions

- ledger ∗ ledger_init ()
- block ∗ new_block ()
- void add_block (block)
- void add_transaction_to_block (block ∗, transaction ∗, int index)
- void add_block_to_ledger (block ∗)
- void find_transaction (struct timespec timestamp, pid_t sender, pid_t receiver)
- void search_timestamp ()
- void search_sender ()
- void search_receiver ()

Variables

- int errno

4.5.1 Macro Definition Documentation

4.5.1.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

Definition at line 28 of file common.h.

4.5.1.2 ARRAY_SIZE

```
#define ARRAY_SIZE(
              x ) (sizeof(x) / sizeof((x)[0]))
```

Definition at line 31 of file common.h.

4.5.1.3 ERROR

```
#define ERROR -1
```

Definition at line 58 of file common.h.

4.5.1.4 EVERYONE_BROKE

```
#define EVERYONE_BROKE '$'
```

Definition at line 46 of file common.h.

4.5.1.5 IPC_ERROR

```
#define IPC_ERROR -1
```

Definition at line 35 of file common.h.

4.5.1.6 LEDGER_ARGV

```
#define LEDGER_ARGV (atoi(argv[4]))
```

Definition at line 52 of file common.h.

4.5.1.7 M_QUEUE_KEY

```
#define M_QUEUE_KEY 0x5AD
```

Definition at line 41 of file common.h.

4.5.1.8 MAX_RETRY

```
#define MAX_RETRY 2
```

Definition at line 57 of file common.h.

4.5.1.9 NODES_PID_ARGV

```
#define NODES_PID_ARGV (atoi(argv[2]))
```

Definition at line 50 of file common.h.

4.5.1.10 NULL

```
#define NULL 0 /* thre's a problem with NULL for some reason */
```

Definition at line 24 of file common.h.

4.5.1.11 PARAMETERS_ARGV

```
#define PARAMETERS_ARGV (atoi(argv[3]))
```

Definition at line 51 of file common.h.

4.5.1.12 RAND

```
#define RAND(
             min,
             max ) ((rand() % (max - min + 1)) + min)
```

Definition at line 32 of file common.h.

4.5.1.13 SELF

```
#define SELF -1
```

Definition at line 45 of file common.h.

4.5.1.14 SEM_ID_ARGV

```
#define SEM_ID_ARGV (atoi(argv[5]))
```

Definition at line 53 of file common.h.

4.5.1.15 SEM_MASTER

```
#define SEM_MASTER 420
```

Definition at line 40 of file common.h.

4.5.1.16 SHM_LEDGER

```
#define SHM_LEDGER 1340
```

Definition at line 39 of file common.h.

4.5.1.17 SHM_NODES_ARRAY

```
#define SHM_NODES_ARRAY 1339
```

Definition at line 38 of file common.h.

4.5.1.18 SHM_PARAMETERS

```
#define SHM_PARAMETERS 1337
```

Definition at line 36 of file common.h.

4.5.1.19 SHM_USERS_ARRAY

```
#define SHM_USERS_ARRAY 1338
```

Definition at line 37 of file common.h.

4.5.1.20 SO_BLOCK_SIZE

```
#define SO_BLOCK_SIZE 100 /* number of transaction per block*/
```

Definition at line 43 of file common.h.

4.5.1.21 SO_REGISTRY_SIZE

```
#define SO_REGISTRY_SIZE 1000 /* max length of consecutive blocks */
```

Definition at line 44 of file common.h.

4.5.1.22 SUCCESS

```
#define SUCCESS 0
```

Definition at line 59 of file common.h.

4.5.1.23 TEST_ERROR

```
#define TEST_ERROR
```

Value:

```
    if (errno)                                     \
    {                                              \
        fprintf(stderr,                            \
                "%s:%d: PID=%5d: Error %d (%s)\n", \
                __FILE__,                          \
                __LINE__,                          \
                getpid(),                          \
                errno,                             \
                strerror(errno));                  \
    }
```

Definition at line 64 of file common.h.

4.5.1.24 USERS_PID_ARGV

```
#define USERS_PID_ARGV (atoi(argv[1]))
```

Definition at line 49 of file common.h.

4.5.1.25 WENT_BROKE

```
#define WENT_BROKE 1
```

Definition at line 56 of file common.h.

4.5.2 Typedef Documentation

4.5.2.1 block

```
typedef struct block_t block
```

4.5.2.2 ledger

```
typedef struct ledger_t ledger
```

4.5.2.3 node

```
typedef struct node_t node
```

4.5.2.4 transaction

```
typedef struct transaction_t transaction
```

4.5.2.5 user

```
typedef struct user_t user
```

4.5.3 Function Documentation

4.5.3.1 add_block()

```
void add_block (
            block  )
```

4.5.3.2 add_block_to_ledger()

```
void add_block_to_ledger (
            block *  )
```

4.5.3.3 add_transaction_to_block()

```
void add_transaction_to_block (
            block * block,
            transaction * newTrans,
            int index )
```

Definition at line 46 of file common.c.

4.5.3.4 find_transaction()

```
void find_transaction (
            struct timespec timestamp,
            pid_t sender,
            pid_t receiver )
```

4.5.3.5 ledger_init()

```
ledger * ledger_init ( )
```

Definition at line 3 of file common.c.

4.5.3.6 new_block()

```
block * new_block ( )
```

Definition at line 24 of file common.c.

4.5.3.7 search_receiver()

```
void search_receiver ( )
```

4.5.3.8 search_sender()

```
void search_sender ( )
```

4.5.3.9 search_timestamp()

```
void search_timestamp ( )
```

4.5.4 Variable Documentation

4.5.4.1 errno

```
int errno  [extern]
```

Definition at line 17 of file sem.c.

4.6 common.h

Go to the documentation of this file.

```c
#ifndef SIMULAZIONE_TRANSAZIONI_COMMON_H
#define SIMULAZIONE_TRANSAZIONI_COMMON_H

#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <sys/types.h>

#include "../utils/debug.h"
#include "../utils/sem.h"
#include "../utils/lists.h"

#ifndef NULL
#define NULL 0 /* thre's a problem with NULL for some reason */
#endif

#ifndef _GNU_SOURCE
#define _GNU_SOURCE
#endif

#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
#define RAND(min, max) ((rand() % (max - min + 1)) + min)

/* -- IPC OBJECTS -- */
#define IPC_ERROR -1
#define SHM_PARAMETERS 1337
#define SHM_USERS_ARRAY 1338
#define SHM_NODES_ARRAY 1339
#define SHM_LEDGER 1340
#define SEM_MASTER 420
#define M_QUEUE_KEY 0x5AD

#define SO_BLOCK_SIZE 100     /* number of transaction per block*/
#define SO_REGISTRY_SIZE 1000 /* max length of consecutive blocks */
#define SELF -1
#define EVERYONE_BROKE '$'

/* -- ARGV LOCATION OF IPC OBJECTS -- */
#define USERS_PID_ARGV (atoi(argv[1]))
#define NODES_PID_ARGV (atoi(argv[2]))
#define PARAMETERS_ARGV (atoi(argv[3]))
#define LEDGER_ARGV (atoi(argv[4]))
#define SEM_ID_ARGV (atoi(argv[5]))

/* -- USER RETURN STATUS -- */
#define WENT_BROKE 1
#define MAX_RETRY 2
#define ERROR -1
#define SUCCESS 0

extern int errno;

#ifndef TEST_ERROR
#define TEST_ERROR                                    \
    if (errno)                                        \
    {                                                 \
        fprintf(stderr,                               \
                "%s:%d: PID=%5d: Error %d (%s)\n",    \
                __FILE__,                             \
                __LINE__,                             \
                getpid(),                             \
                errno,                                \
                strerror(errno));                     \
    }
#endif

struct parameters
{
    unsigned int SO_USER_NUM;
    unsigned int SO_NODES_NUM;
    unsigned int SO_BUDGET_INIT;
    char SO_REWARD; /* max value 100 */
    unsigned long SO_MIN_TRANS_GEN_NSEC;
    unsigned long SO_MAX_TRANS_GEN_NSEC;
    unsigned int SO_RETRY;
    unsigned int SO_TP_SIZE;
    unsigned long SO_MIN_TRANS_PROC_NSEC;
    unsigned long SO_MAX_TRANS_PROC_NSEC;
    unsigned int SO_SIM_SEC;
    unsigned int SO_FRIENDS_NUM;
    unsigned int SO_HOPS;
};

typedef struct user_t
{
    pid_t pid;
    enum
    {
        alive,
        broke,
        dead
    } status;
} user;

typedef struct node_t
{
    pid_t pid;
    enum
    {
        available,
        full
    } status;
} node;

/* Transaction struct */
typedef struct transaction_t
{
    struct timespec timestamp;
    pid_t sender;
    pid_t receiver;
    int amount;
    int reward;
    enum
    {
        pending,
        processing,
        confirmed,
        aborted
    } status;
} transaction;

struct message
{
    long mtype;
    transaction userTrans;
};

/* Block struct */
typedef struct block_t
{
    transaction transList[SO_BLOCK_SIZE];
    unsigned int blockIndex; /* when a block is written on ledger it's Index needs to be updated */
    struct block *next;
    struct block *prev;
} block;

/* Libro Mastro (ledger) struct */
typedef struct ledger_t
{
    block *head;
    unsigned int registryCurrSize; /* initialize to SO_REGISTRY_SIZE, update with every new block
 added */
} ledger;

ledger *ledger_init();
block *new_block();
void add_block(block);
void add_transaction_to_block(block *, transaction *, int index);
void add_block_to_ledger(block *);
void find_transaction(struct timespec timestamp, pid_t sender, pid_t receiver); /* NULL used to group
 results */

/* listparser.c */
void search_timestamp();
void search_sender();
void search_receiver();

#endif /* SIMULAZIONE_TRANSAZIONI_COMMON_H */
```

4.7 src/include/master.h File Reference

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

Include dependency graph for master.h: This graph shows which files directly or indirectly include this file:

Functions

- void make_arguments (int ∗IPCarray, char ∗argv[ ])
- pid_t spawn_user (char ∗argv[ ])
- pid_t spawn_node (char ∗argv[ ])
- void shared_memory_objects_init (int ∗shared_memory_objects_IDs)
- void semaphores_init ()
- void make_ipc_array (int ∗IPC_objects_IDs)
- void master_interrupt_handle (int signum)

4.7.1 Function Documentation

4.7.1.1 make_arguments()

```
void make_arguments (
            int * IPCarray,
            char * argv[ ] )
```

4.7.1.2 make_ipc_array()

```
void make_ipc_array (
            int * IPC_objects_IDs )
```

Definition at line 175 of file master.c.

4.7.1.3 master_interrupt_handle()

```
void master_interrupt_handle (
            int signum )
```

Definition at line 188 of file master.c.

4.7.1.4 semaphores_init()

```
void semaphores_init ( )
```

Definition at line 167 of file master.c.

4.7.1.5 shared_memory_objects_init()

```
void shared_memory_objects_init (
             int * shared_memory_objects_IDs )
```

Definition at line 111 of file master.c.

4.7.1.6 spawn_node()

```
pid_t spawn_node (
             char * argv[ ] )
```

Definition at line 88 of file master.c.

4.7.1.7 spawn_user()

```
pid_t spawn_user (
             char * argv[ ] )
```

Definition at line 65 of file master.c.

4.8 master.h

Go to the documentation of this file.

```
#ifndef SIMULAZIONE_TRANSAZIONI_MASTER_H
#define SIMULAZIONE_TRANSAZIONI_MASTER_H

/*
 * probabilmente tutte set macro non ci servono perche' ha più senso passare i
 * valori come parametri in compilazione
 */

/* libro mastro ----> linked list */

/*
 * execve(const char *user, char *const argv[], char *const envp[])
 * in user_fork to link "user" executable to the forked process, same thing for nodes
 * check lesson on pipes by prof. bini, the second hour
 */

/*
 * every transaction should be noted and we need to manage inconsistency with
 * semaphores maybe
 */

```

```
/*
 * get every user_pid, ask to libro_mastro
 * to return it's current_budget and print
 * repeat every second until simulation persists
 * remember that CTRL-C should kill the simulation
 */

/*
 * need to define a kill signal for the simulation, either:
 * - SO_SIM_SEC seconds have passed
 * - libro_mastro is full
 * - CTRL-C from stdin
 */

/*
 * end of simulation should print:
 * - kill signal
 * - balance of every user, as before, may need to write a funciton for that
 * - balance of every node (function as before but with different parameter)
 * - number of user processes aborted
 * - number of blocks in the libro_mastro
 * - number of transaction still in the pool, for each node
 */

#include <string.h>
#include <signal.h>

void make_arguments(int *IPCarray, char *argv[]);

pid_t spawn_user(char *argv[]);
pid_t spawn_node(char *argv[]);

void shared_memory_objects_init(int *shared_memory_objects_IDs);
void semaphores_init();
void make_ipc_array(int *IPC_objects_IDs);

void master_interrupt_handle(int signum);

#endif /* SIMULAZIONE_TRANSAZIONI_MASTER_H */
```

4.9 src/include/nodes.h File Reference

This graph shows which files directly or indirectly include this file:

Variables

- struct timespec randSleepTime
- struct timespec sleepTimeRemaining

4.9.1 Variable Documentation

4.9.1.1 randSleepTime

```
struct timespec randSleepTime
```

Definition at line 1 of file nodes.h.

4.9.1.2 sleepTimeRemaining

```
struct timespec sleepTimeRemaining
```

Definition at line 2 of file nodes.h.

4.10 nodes.h

Go to the documentation of this file.

```
00001 struct timespec randSleepTime;
00002 struct timespec sleepTimeRemaining;
```

4.11 src/include/parser.h File Reference

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

Include dependency graph for parser.h: This graph shows which files directly or indirectly include this file:

Macros

- #define NUM_PARAMETERS 13
- #define CONF_FILE "conf.txt"
- #define CONF_ERROR -1

Functions

- int parse_parameters (struct parameters ∗par)
- void assign_defaults (struct parameters ∗par)

4.11.1 Macro Definition Documentation

4.11.1.1 CONF_ERROR

```
#define CONF_ERROR -1
```

Definition at line 9 of file parser.h.

4.11.1.2 CONF_FILE

```
#define CONF_FILE "conf.txt"
```

Definition at line 8 of file parser.h.

4.11.1.3 NUM_PARAMETERS

```
#define NUM_PARAMETERS 13
```

Definition at line 7 of file parser.h.

4.11.2 Function Documentation

4.11.2.1 assign_defaults()

```
void assign_defaults (
             struct parameters * par )
```

Definition at line 41 of file parser.c.

4.11.2.2 parse_parameters()

```
int parse_parameters (
             struct parameters * par )
```

Definition at line 58 of file parser.c.

4.12 parser.h

Go to the documentation of this file.

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_PARSER_H
00002 #define SIMULAZIONE_TRANSAZIONI_PARSER_H
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006
00007 #define NUM_PARAMETERS 13
00008 #define CONF_FILE "conf.txt"
00009 #define CONF_ERROR -1
00010
00011 int parse_parameters(struct parameters *par);
00012
00013 void assign_defaults(struct parameters *par);
00014
00015 #endif /* SIMULAZIONE_TRANSAZIONI_PARSER_H */
```

4.13 src/include/print.h File Reference

```
#include "common.h"
```

Include dependency graph for print.h: This graph shows which files directly or indirectly include this file:

Functions

- void print_time_to_die ()
- void print_user_nodes_table (pid_t main, user ∗user, node ∗nodes, struct parameters ∗par)
- void print_kill_signal ()
- void print_user_balance ()
- void print_node_balance ()
- void print_num_aborted ()
- void print_num_blocks ()
- void print_transactions_still_in_pool ()
- void final_print (pid_t masterPID, user ∗usersPID, node ∗nodesPID, struct parameters ∗par)
- void print_parameters (struct parameters ∗par)
- void print_block (FILE ∗fp, block ∗b)
- void print_transaction (FILE ∗fp, transaction ∗t)
- void print_ledger (ledger ∗l)
- void formatted_timestamp (FILE ∗fp)

4.13.1 Function Documentation

4.13.1.1 final_print()

```
void final_print (
             pid_t masterPID,
             user * usersPID,
             node * nodesPID,
             struct parameters * par )
```

Definition at line 40 of file print.c.

4.13.1.2 formatted_timestamp()

```
void formatted_timestamp (
             FILE * fp )
```

Definition at line 157 of file print.c.

4.13.1.3 print_block()

```
void print_block (
             FILE * fp,
             block * b )
```

Definition at line 127 of file print.c.

4.13.1.4 print_kill_signal()

```
void print_kill_signal ( )
```

4.13.1.5 print_ledger()

```
void print_ledger (
             ledger * l )
```

Definition at line 143 of file print.c.

4.13.1.6 print_node_balance()

```
void print_node_balance ( )
```

4.13.1.7 print_num_aborted()

```
void print_num_aborted ( )
```

4.13.1.8 print_num_blocks()

```
void print_num_blocks ( )
```

4.13.1.9 print_parameters()

```
void print_parameters (
             struct parameters * par )
```

Definition at line 52 of file print.c.

4.13.1.10 print_time_to_die()

```
void print_time_to_die ( )
```

Definition at line 7 of file print.c.

4.13.1.11 print_transaction()

```
void print_transaction (
            FILE * fp,
            transaction * t )
```

Definition at line 98 of file print.c.

4.13.1.12 print_transactions_still_in_pool()

```
void print_transactions_still_in_pool ( )
```

4.13.1.13 print_user_balance()

```
void print_user_balance ( )
```

4.13.1.14 print_user_nodes_table()

```
void print_user_nodes_table (
            pid_t main,
            user * user,
            node * nodes,
            struct parameters * par )
```

Definition at line 12 of file print.c.

4.14 print.h

Go to the documentation of this file.

```
#ifndef SIMULAZIONE_TRANSAZIONI_PRINT_H
#define SIMULAZIONE_TRANSAZIONI_PRINT_H

#include "common.h"

void print_time_to_die();
void print_user_nodes_table(pid_t main, user *user, node *nodes, struct parameters *par); /* function that prints on terminal the PID of every user and node process */
void print_kill_signal();                                                              /* need to define, prints reason of termination (simTime elapsed/ledger full/every process terminated) */
void print_user_balance();                                                             /* need to define, prints balance of every user */
void print_node_balance();                                                             /* need to define, prints balance of every node */
void print_num_aborted();                                                              /* need to define, prints num of processes aborted */
void print_num_blocks();                                                               /* need to define, prints num of blocks saved in the ledger */
void print_transactions_still_in_pool();                                               /* need to define, prints num of transactions still in the pool of each node */

void final_print(pid_t masterPID, user *usersPID, node *nodesPID, struct parameters *par);
void print_parameters(struct parameters *par);

/* formatting ledger and blocks */
void print_block(FILE *fp, block *b);
void print_transaction(FILE *fp, transaction *t);
void print_ledger(ledger *l);

void formatted_timestamp(FILE *fp);

#endif /* SIMULAZIONE_TRANSAZIONI_PRINT_H */
```

4.15 src/include/users.h File Reference

```
#include <signal.h>
#include <stdlib.h>
#include <time.h>
```

Include dependency graph for users.h: This graph shows which files directly or indirectly include this file:

Macros

- #define SLEEP_TIME_SET
- #define SLEEP

Functions

- void user_transactions_handle (int signum)
- void user_interrupt_handle (int signum)
- int get_pid_userIndex (int PID_toSearch)
- pid_t get_random_userPID ()
- pid_t get_random_nodePID ()
- void queue_to_pid (pid_t nodePID)
- void transaction_init (pid_t nodePID, int amount, int reward)
- int send_transaction ()

4.15.1 Macro Definition Documentation

4.15.1.1 SLEEP

```
#define SLEEP
```

Value:

```
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &randSleepTime, &sleepTimeRemaining); \
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &sleepTimeRemaining, NULL);
```

Definition at line 14 of file users.h.

4.15.1.2 SLEEP_TIME_SET

```
#define SLEEP_TIME_SET
```

Value:

```
    randSleepTime.tv_sec = 0; \
    randSleepTime.tv_nsec = RAND(par->SO_MIN_TRANS_GEN_NSEC, par->SO_MAX_TRANS_GEN_NSEC);
```

Definition at line 10 of file users.h.

4.15.2 Function Documentation

4.15.2.1 get_pid_userIndex()

```
int get_pid_userIndex (
             int PID_toSearch )
```

Definition at line 51 of file users.c.

4.15.2.2 get_random_nodePID()

```
pid_t get_random_nodePID ( )
```

Definition at line 83 of file users.c.

4.15.2.3 get_random_userPID()

```
pid_t get_random_userPID ( )
```

Definition at line 65 of file users.c.

4.15.2.4 queue_to_pid()

```
void queue_to_pid (
             pid_t nodePID )
```

Definition at line 138 of file users.c.

4.15.2.5 send_transaction()

```
int send_transaction ( )
```

Definition at line 176 of file users.c.

4.15.2.6 transaction_init()

```
void transaction_init (
             pid_t nodePID,
             int amount,
             int reward )
```

Definition at line 146 of file users.c.

4.15.2.7 user_interrupt_handle()

```
void user_interrupt_handle (
             int signum )
```

Definition at line 227 of file users.c.

4.15.2.8 user_transactions_handle()

```
void user_transactions_handle (
             int signum )
```

Definition at line 217 of file users.c.

4.16 users.h

Go to the documentation of this file.

```
#ifndef SIMULAZIONE_TRANSAZIONI_USERS_H
#define SIMULAZIONE_TRANSAZIONI_USERS_H

#include <signal.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>

/* sets sleep time with nsec precision for trans_gen */
#define SLEEP_TIME_SET        \
    randSleepTime.tv_sec = 0; \
    randSleepTime.tv_nsec = RAND(par->SO_MIN_TRANS_GEN_NSEC, par->SO_MAX_TRANS_GEN_NSEC);

#define SLEEP                                                                      \
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &randSleepTime, &sleepTimeRemaining); \
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &sleepTimeRemaining, NULL);


void user_transactions_handle(int signum);
void user_interrupt_handle(int signum);

int get_pid_userIndex(int PID_toSearch);
pid_t get_random_userPID();
pid_t get_random_nodePID();

void queue_to_pid(pid_t nodePID);
void transaction_init(pid_t nodePID, int amount, int reward);
int send_transaction();

#endif /* SIMULAZIONE_TRANSAZIONI_USERS_H */
```

4.17 src/master.c File Reference

```
#include "include/common.h"
#include "include/master.h"
#include "include/print.h"
#include "include/parser.h"
```
Include dependency graph for master.c:

Macros

- #define SHM_NUM 4
- #define SEM_NUM 1
- #define IPC_NUM 8
- #define USER_NAME "./users"
- #define NODE_NAME "./nodes"

Functions

- void make_arguments (int ∗IPC_array, char ∗∗argv)
- pid_t spawn_user (char ∗userArgv[ ])
- pid_t spawn_node (char ∗nodeArgv[ ])
- void shared_memory_objects_init (int ∗shmArray)
- void semaphores_init ()
- void make_ipc_array (int ∗IPC_array)
- void master_interrupt_handle (int signum)
- int main (int argc, char ∗argv[ ])

Variables

- struct parameters ∗ par
- user ∗ usersPID
- node ∗ nodesPID
- ledger ∗ mainLedger
- int semID

4.17.1 Macro Definition Documentation

4.17.1.1 IPC_NUM

```
#define IPC_NUM 8
```

Definition at line 8 of file master.c.

4.17.1.2 NODE_NAME

```
#define NODE_NAME "./nodes"
```

Definition at line 11 of file master.c.

4.17.1.3 SEM_NUM

```
#define SEM_NUM 1
```

Definition at line 7 of file master.c.

4.17.1.4 SHM_NUM

```
#define SHM_NUM 4
```

Definition at line 6 of file master.c.

4.17.1.5 USER_NAME

```
#define USER_NAME "./users"
```

Definition at line 10 of file master.c.

4.17.2 Function Documentation

4.17.2.1 main()

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

Definition at line 210 of file master.c.

4.17.2.2 make_arguments()

```
void make_arguments (
             int * IPC_array,
             char ** argv )
```

Definition at line 36 of file master.c.

4.17.2.3 make_ipc_array()

```
void make_ipc_array (
             int * IPC_array )
```

Definition at line 175 of file master.c.

4.17.2.4 master_interrupt_handle()

```
void master_interrupt_handle (
             int signum )
```

Definition at line 188 of file master.c.

4.17.2.5 semaphores_init()

```
void semaphores_init ( )
```

Definition at line 167 of file master.c.

4.17.2.6 shared_memory_objects_init()

```
void shared_memory_objects_init (
             int * shmArray )
```

Definition at line 111 of file master.c.

4.17.2.7 spawn_node()

```
pid_t spawn_node (
             char * nodeArgv[ ] )
```

Definition at line 88 of file master.c.

4.17.2.8 spawn_user()

```
pid_t spawn_user (
            char * userArgv[] )
```

Definition at line 65 of file master.c.

4.17.3 Variable Documentation

4.17.3.1 mainLedger

```
ledger* mainLedger
```

Definition at line 23 of file master.c.

4.17.3.2 nodesPID

```
node* nodesPID
```

Definition at line 22 of file master.c.

4.17.3.3 par

```
struct parameters* par
```

Definition at line 20 of file master.c.

4.17.3.4 semID

```
int semID
```

Definition at line 25 of file master.c.

4.17.3.5 usersPID

```
user* usersPID
```

Definition at line 21 of file master.c.

4.18 master.c

Go to the documentation of this file.

```
#include "include/common.h"
#include "include/master.h"
#include "include/print.h"
#include "include/parser.h"

#define SHM_NUM 4
#define SEM_NUM 1
#define IPC_NUM 8

#define USER_NAME "./users"
#define NODE_NAME "./nodes"

/*
 ======================
 || GLOBAL VARIABLES ||
 ======================
 */

/* parameters of simulation */
struct parameters *par;
user *usersPID;
node *nodesPID;
ledger *mainLedger;

int semID;

/*extern int usersPrematurelyDead = 0;*/

/*
 ======================
 ||    FUNCTIONS     ||
 ======================
 */

/* make argv array with IPC IDs for user and nodes */
void make_arguments(int *IPC_array, char **argv)
{
    char *uPID_array = malloc(3 * sizeof(IPC_array[0]) + 1);
    char *nPID_array = malloc(3 * sizeof(IPC_array[0]) + 1);
    char *parameters = malloc(3 * sizeof(IPC_array[0]) + 1);
    char *ledger = malloc(3 * sizeof(IPC_array[0]) + 1);
    char *semID = malloc(3 * sizeof(IPC_array[0]) + 1);

    sprintf(uPID_array, "%d", IPC_array[0]);
    sprintf(nPID_array, "%d", IPC_array[1]);
    sprintf(parameters, "%d", IPC_array[2]);
    sprintf(ledger, "%d", IPC_array[3]);
    sprintf(semID, "%d", IPC_array[4]);

    argv[0] = USER_NAME; /* need nodes to have a different name but not a priority */
    argv[1] = uPID_array;
    TRACE((":master: argv[uPID] = %s\n", uPID_array))
    argv[2] = nPID_array;
    TRACE((":master: argv[nPID] = %s\n", nPID_array))
    argv[3] = parameters;
    TRACE((":master: argv[par] = %s\n", parameters))
    argv[4] = ledger;
    TRACE((":master: argv[ledger] = %s\n", ledger))
    argv[5] = semID;
    TRACE((":master: argv[sem] = %s\n", semID))
    argv[8] = NULL; /* Terminating argv with NULL value */
}

/* fork and execve a "./users" */
pid_t spawn_user(char *userArgv[])
{
    pid_t myPID = fork();
    TRACE((":master: argv values: %s %s %s %s %s %s\n", userArgv[0], userArgv[1], userArgv[2],
 userArgv[3], userArgv[4], userArgv[5]))
    switch (myPID)
    {
    case -1: /* Error case */
        printf("-- Error forking for user\n");
        break;

    case 0: /* Child case */
        TRACE((":master: Spawning user\n"));
        execve(USER_NAME, userArgv, NULL);
        TEST_ERROR
        TRACE(("!! Message that should never be seen\n"));
        break;

    default:
        return myPID;
    }
}

/* fork and execve a "./nodes" */
pid_t spawn_node(char *nodeArgv[])
{
    pid_t myPID = fork();
    TRACE((":master: argv values: %s %s %s %s %s %s\n", nodeArgv[0], nodeArgv[1], nodeArgv[2],
 nodeArgv[3], nodeArgv[4], nodeArgv[5]))
    switch (myPID)
    {
    case -1: /* Error case */
        printf("!! Error forking for node\n");
        break;

    case 0: /* Child case */
        TRACE((":master: Spawning node\n"));
        execve(NODE_NAME, nodeArgv, NULL);
        TEST_ERROR
        TRACE(("!! Message that should never be seen\n"));
        break;

    default:
        return myPID;
    }
}

/* attach usersPID, nodesPID, par and mainLedger to shared memory, returns an array with respective
 IDs */
void shared_memory_objects_init(int *shmArray)
{
    /* shared memory segments IDs */
    int usersPID_ID;
    int nodesPID_ID;
    int mainLedger_ID;
    int par_ID;

    /* parameters init and read from conf file */
    par_ID = shmget(SHM_PARAMETERS, sizeof(par), 0600 | IPC_CREAT | IPC_EXCL);
    TEST_ERROR
    par = (struct parameters *)shmat(par_ID, NULL, 0);
    if (parse_parameters(par) == CONF_ERROR)
    {
        TRACE(("-- Error reading conf file, defaulting to conf#1\n"));
    }
    else
    {
        TRACE(("-- Conf file read successful\n"));
    }
#ifdef DEBUG
    print_parameters(par);
#endif
    /* (users_t) and (nodes_t) arrays */
    usersPID_ID = shmget(SHM_USERS_ARRAY,
                         (par->SO_USER_NUM) * sizeof(user),
                         0600 | IPC_CREAT | IPC_EXCL);
    TEST_ERROR
    nodesPID_ID = shmget(SHM_NODES_ARRAY,
                         (par->SO_NODES_NUM) * sizeof(node),
                         0600 | IPC_CREAT | IPC_EXCL);
    TEST_ERROR
    usersPID = (user *)shmat(usersPID_ID, NULL, 0);
    nodesPID = (node *)shmat(nodesPID_ID, NULL, 0);

    /* mainLedger */
    mainLedger_ID = shmget(SHM_LEDGER,
                           (par->SO_NODES_NUM) * sizeof(node),
                           0600 | IPC_CREAT | IPC_EXCL);
    TEST_ERROR
    mainLedger = (ledger *)shmat(mainLedger_ID, NULL, 0);

    /* mark for deallocation so that they are automatically
     * removed once master dies
     */
    shmctl(usersPID_ID, IPC_RMID, NULL);
    shmctl(nodesPID_ID, IPC_RMID, NULL);
    shmctl(par_ID, IPC_RMID, NULL);
    shmctl(mainLedger_ID, IPC_RMID, NULL);

    shmArray[0] = usersPID_ID;
    shmArray[1] = nodesPID_ID;
    shmArray[2] = par_ID;
    shmArray[3] = mainLedger_ID;
}

```

```c
void semaphores_init()
{
    semID = semget(SEM_MASTER, 1, 0600 | IPC_CREAT | IPC_EXCL);
    TEST_ERROR
    TRACE((":master: semID is %d\n", semID));
}

/* makes an array with every IPC object ID */
void make_ipc_array(int *IPC_array)
{
    int shmIDs[SHM_NUM]; /* array containing every shared memory ID */
    int semIDs[1] = {0};

    shared_memory_objects_init(shmIDs);
    semIDs[0] = semID;
    /* semaphores_init(semIDs); */
    memcpy(IPC_array, shmIDs, SHM_NUM * sizeof(int));
    memcpy(IPC_array + SHM_NUM, shmIDs, SEM_NUM * sizeof(int));
}

/* CTRL-C handler */
void master_interrupt_handle(int signum)
{
    write(1, "::Master:: SIGINT ricevuto\n", 28);
    killpg(0, SIGINT);

    /* just to avoid printing before everyone has finished*/
    sleep(1);
    final_print(getpid(), usersPID, nodesPID, par);

    /*
    int status;

    while (wait(&status) != -1)
    {
        status » 8; /* no idea about what it does please help *
    }
    */

    semctl(semID, 1, IPC_RMID);
    exit(0);
}

int main(int argc, char *argv[])
{
    pid_t myPID = getpid();

    int uCounter, nCounter, returnVal;
    int simTime;
    int ipcObjectsIDs[IPC_NUM];
    char **argvSpawns = malloc(8*32);

    struct sigaction sa;
    struct sembuf sops;

    semaphores_init();
    make_ipc_array(ipcObjectsIDs);
    make_arguments(ipcObjectsIDs, argvSpawns);
    mainLedger = ledger_init();

    simTime = par->SO_SIM_SEC;

    /* -- SIGNAL HANDLER --
     * first set all bytes of sigation to 0
     * then initialize sa.handler to a pointer to the function interrupt_handle
     * then set the handler to handle SIGINT signals ((struct sigaction *oldact) = NULL)
     */
    bzero(&sa, sizeof(sa));
    sa.sa_handler = master_interrupt_handle;
    sigaction(SIGINT, &sa, NULL);

    for (nCounter = 0; nCounter < par->SO_NODES_NUM; nCounter++)
    {
        LOCK
            nodesPID[nCounter]
                .status = available;
        nodesPID[nCounter].pid = spawn_node(argvSpawns);
        UNLOCK
        if (getpid() != myPID)
        {
            return;
        }
    }

    /*usersPrematurelyDead = 0;*/
    for (uCounter = 0; uCounter < par->SO_USER_NUM; uCounter++)
    {
        LOCK
            usersPID[uCounter]
                .status = alive;
        usersPID[uCounter].pid = spawn_user(argvSpawns);
        UNLOCK
        if (getpid() != myPID)
        {
            switch (returnVal = wait(NULL))
            {
            case MAX_RETRY:
                /* change status in usersPID */
                printf("User %d has died because of too many retries :(\n", getpid());
                break;
            }

            return;
        }
    }

    sleep(simTime);

    print_time_to_die();
    killpg(0, SIGINT); /* our sigint handler needs to do quite a lot of things to print the wall of test below */

    return 0;
}
```

4.19 src/nodes.c File Reference

```
#include <time.h>
#include "include/nodes.h"
#include "include/common.h"
```

Include dependency graph for nodes.c:

Functions

- int main (int argc, char ∗argv[ ])

Variables

- struct parameters ∗ par
- user ∗ usersPID
- node ∗ nodesPID
- ledger ∗ mainLedger
- int semID
- int queueID
- pid_t myPID

4.19.1 Function Documentation

4.19.1.1 main()

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

Definition at line 58 of file nodes.c.

4.19.2 Variable Documentation

4.19.2.1 mainLedger

```
ledger* mainLedger
```

Definition at line 17 of file nodes.c.

4.19.2.2 myPID

```
pid_t myPID
```

Definition at line 22 of file nodes.c.

4.19.2.3 nodesPID

```
node* nodesPID
```

Definition at line 16 of file nodes.c.

4.19.2.4 par

```
struct parameters* par
```

Definition at line 14 of file nodes.c.

4.19.2.5 queueID

```
int queueID
```

Definition at line 20 of file nodes.c.

4.19.2.6 semID

```
int semID
```

Definition at line 19 of file nodes.c.

4.19.2.7 usersPID

```
user* usersPID
```

Definition at line 15 of file nodes.c.

4.20 nodes.c

Go to the documentation of this file.

```
00001 #include <time.h>
00002 #include "include/nodes.h"
00003 #include "include/common.h"
00004
00005 /* transaction pool==transaction's array */
00006
00007 /*
00008  ======================
00009  || GLOBAL VARIABLES ||
00010  ======================
00011  */
00012
00013 /* parameters of simulation */
00014 struct parameters *par;
00015 user *usersPID;
00016 node *nodesPID;
00017 ledger *mainLedger;
00018
00019 int semID;
00020 int queueID;
00021
00022 pid_t myPID;
00023
00024 /*void Node()
00025 {
00026     int t_pool[SO_TP_SIZE];
00027     checkTpFull(t_pool[SO_TP_SIZE]);
00028     arrayProcesser();
00029     createBlock();
00030 }
00031
00032 int checkTpFull(int t_pool[SO_TP_SIZE])
00033 {
00034     if (t_pool[SO_TP_SIZE] == SO_TP_SIZE)
00035     {
00036         return 0;
00037     }
00038 }
00039
00040 void arrayProcesser()
00041 {
00042     int i = 0;
00043     for (i; i < SO_TP_SIZE - 1; i++)
00044     {
00045     }
00046 }
00047
00048 int createBlock()
00049 {
00050 }
00051
00052 int sleepMethod(int argc, char *argv[])
00053 {
00054     randSleepTime.tv_sec = 0;
00055     randSleepTime.tv_nsec = RAND(SO_MIN_TRANS_PROC_NSEC, SO_MAX_TRANS_PROC_NSEC);
}*/

int main(int argc, char *argv[])
{
    int myPID = getpid();
    printf("Node %d has finished\n", myPID);
    return 0;

    queueID = msgget(myPID, IPC_CREAT | 0600);
}
```

4.21 src/parser.c File Reference

```
#include "include/common.h"
#include "utils/debug.h"
#include "include/parser.h"
```

Include dependency graph for parser.c:

Functions

- void assign_defaults (struct parameters *par)
- int parse_parameters (struct parameters *par)

4.21.1 Function Documentation

4.21.1.1 assign_defaults()

```
void assign_defaults (
            struct parameters * par )
```

Definition at line 41 of file parser.c.

4.21.1.2 parse_parameters()

```
int parse_parameters (
            struct parameters * par )
```

Definition at line 58 of file parser.c.

4.22 parser.c

Go to the documentation of this file.

```
#include "include/common.h"
#include "utils/debug.h"
#include "include/parser.h"

/*enum paramID
{
    SO_USER_NUM,
    SO_NODES_NUM,
    SO_NUM_FRIENDS,
    SO_SIM_SEC,
    SO_HOPS,
    SO_BUDGET_INIT,
    SO_REWARD,
    SO_MIN_TRANS_GEN_NSEC,
    SO_MAX_TRANS_GEN_NSEC,
    SO_RETRY,
    SO_TP_SIZE,
    SO_MIN_TRANS_PROC_NSEC,
    SO_MAX_TRANS_PROC_NSEC
};

struct parameters
{
    char *string;
    enum paramID id;
} paramList[] = {
    {"SO_USER_NUM", SO_USER_NUM},
    {"SO_NODES_NUM", SO_NODES_NUM},
    {"SO_NUM_FRIENDS", SO_NUM_FRIENDS},
    {"SO_SIM_SEC", SO_SIM_SEC},
    {"SO_HOPS", SO_HOPS},
    {"SO_BUDGET_INIT", SO_BUDGET_INIT},
    {"SO_REWARD", SO_REWARD},
    {"SO_MIN_TRANS_GEN_NSEC", SO_MIN_TRANS_GEN_NSEC},
    {"SO_MAX_TRANS_GEN_NSEC", SO_MAX_TRANS_GEN_NSEC},
    {"SO_RETRY", SO_RETRY},
    {"SO_TP_SIZE", SO_TP_SIZE},
    {"SO_MIN_TRANS_PROC_NSEC", SO_MIN_TRANS_PROC_NSEC},
    {"SO_MAX_TRANS_PROC_NSEC", SO_MAX_TRANS_PROC_NSEC}};*/

void assign_defaults(struct parameters *par)
{
    par->SO_USER_NUM = 100;
    par->SO_NODES_NUM = 10;
    par->SO_BUDGET_INIT = 1000;
    par->SO_REWARD = 1;
    par->SO_MIN_TRANS_GEN_NSEC = 100000000;
    par->SO_MAX_TRANS_GEN_NSEC = 200000000;
    par->SO_RETRY = 20;
    par->SO_TP_SIZE = 1000;
    par->SO_MIN_TRANS_PROC_NSEC = 100000000;
    par->SO_MAX_TRANS_PROC_NSEC = 200000000;
    par->SO_SIM_SEC = 10;
    par->SO_FRIENDS_NUM = 3;
    par->SO_HOPS = 10;
}

int parse_parameters(struct parameters *par)
{
    FILE *fp;

    /*enum paramID tokensE;*/

    /* longer than NUM_PARAMETERS to account for comments and such */
    char buffer[128];
    int i = 0;

    char *tokens[NUM_PARAMETERS];
    unsigned long values[NUM_PARAMETERS]; /* downcast is easy, upcast not so much */

    /*struct parameters *par = malloc(sizeof(struct parameters));*/
    assign_defaults(par);
    TRACE((":parser: assigned defaults\n"));

    fp = fopen(CONF_FILE, "r");
    if (fp == NULL)
        return CONF_ERROR; /* default config */

    while (fgets(buffer, 127, fp))
    {
        tokens[i] = malloc(64);
        /*values[i] = malloc(sizeof(int));*/

        sscanf(buffer, "%s %lu", tokens[i], &values[i]);

        i++;
    }

    for (i = 0; i < NUM_PARAMETERS; i++)
    {
        /*switch(tokensE){
            case SO_USER_NUM:
            case SO_NODES_NUM:
            case SO_NUM_FRIENDS:
            case SO_SIM_SEC:
            case SO_HOPS:
            case SO_BUDGET_INIT:
            case SO_REWARD:
            case SO_MIN_TRANS_GEN_NSEC:
            case SO_MAX_TRANS_GEN_NSEC:
            case SO_RETRY:
            case SO_TP_SIZE:
            case SO_MIN_TRANS_PROC_NSEC:
            case SO_MAX_TRANS_PROC_NSEC:
            default:
            break;
        } it can be implemented in a nicer to look at way, but not now */

        /*printf("%s\n",tokens[i]);*/

        if (!strcmp(tokens[i], "SO_USER_NUM"))
        {
            par->SO_USER_NUM = values[i];
        }
        else if (!strcmp(tokens[i], "SO_NODES_NUM"))
        {
            par->SO_NODES_NUM = values[i];
        }
        else if (!strcmp(tokens[i], "SO_BUDGET_INIT"))
        {
            par->SO_BUDGET_INIT = values[i];
        }
        else if (!strcmp(tokens[i], "SO_REWARD"))
        {
            /* given that it is a char it's very easy to get it out of bound,
             * I prefer straight up normalizing it rather than resetting everything
             * because of ERANGE
             */
            if (values[i] >= 0 && values[i] <= 100)
                par->SO_REWARD = values[i];
            else
                printf(":parser: SO_REWARD incorrect value, resetting default\n");
        }
        else if (!strcmp(tokens[i], "SO_MIN_TRANS_GEN_NSEC"))
        {
            par->SO_MIN_TRANS_GEN_NSEC = values[i];
        }
        else if (!strcmp(tokens[i], "SO_MAX_TRANS_GEN_NSEC"))
        {
            par->SO_MAX_TRANS_GEN_NSEC = values[i];
        }
        else if (!strcmp(tokens[i], "SO_RETRY"))
        {
            par->SO_RETRY = values[i];
        }
        else if (!strcmp(tokens[i], "SO_TP_SIZE"))
        {
            par->SO_TP_SIZE = values[i];
        }
        else if (!strcmp(tokens[i], "SO_MIN_TRANS_PROC_NSEC"))
        {
            par->SO_MIN_TRANS_PROC_NSEC = values[i];
        }
        else if (!strcmp(tokens[i], "SO_MAX_TRANS_PROC_NSEC"))
        {
            par->SO_MAX_TRANS_PROC_NSEC = values[i];
        }
        else if (!strcmp(tokens[i], "SO_SIM_SEC"))
        {
            par->SO_SIM_SEC = values[i];
        }
        else if (!strcmp(tokens[i], "SO_FRIENDS_NUM"))
        {
            par->SO_FRIENDS_NUM = values[i];
        }
        else if (!strcmp(tokens[i], "SO_HOPS"))
        {
            par->SO_HOPS = values[i];
        }
    }

    /* -- CONF ERRORS CORRECTION -- */
    if (errno == ERANGE)
    {
        TRACE((":parser: one or multiple values out of bound, resetting defaults\n"));
        assign_defaults(par);
    }
    if (par->SO_MIN_TRANS_GEN_NSEC > par->SO_MAX_TRANS_GEN_NSEC)
    {
        TRACE((":parser: SO_MIN_TRANS_GEN_NSEC greater than SO_MAX_TRANS_GEN_NSEC, will be
normalized\n"));
        par->SO_MIN_TRANS_GEN_NSEC = par->SO_MAX_TRANS_GEN_NSEC;
    }
    if (par->SO_MIN_TRANS_PROC_NSEC > par->SO_MAX_TRANS_PROC_NSEC)
    {
        TRACE((":parser: SO_MIN_TRANS_PROC_NSEC greater than SO_MAX_TRANS_PROC_NSEC, will be
normalized\n"));
        par->SO_MIN_TRANS_PROC_NSEC = par->SO_MAX_TRANS_PROC_NSEC;
    }

    TRACE(("--------------------------------------------\n----------- Configuration input
------------\n"));
    for (i = 0; i < NUM_PARAMETERS; i++)
    {
        TRACE(("%s %lu\n", tokens[i], values[i]));
        free(tokens[i]);
    }
    TRACE(("--------------------------------------------\n"));

    fclose(fp);

    return 0;
}
```

4.23 src/print.c File Reference

```
#include "include/common.h"
#include "include/print.h"
```

Include dependency graph for print.c:

Functions

- void print_time_to_die ()
- void print_user_nodes_table (pid_t mainPID, user ∗userPID, node ∗nodePID, struct parameters ∗par)
- void print_kill_signal ()
- void print_user_balance ()
- void print_node_balance ()
- void print_num_aborted ()
- void print_num_blocks ()
- void print_transactions_still_in_pool ()
- void final_print (pid_t masterPID, user ∗usersPID, node ∗nodesPID, struct parameters ∗par)
- void print_parameters (struct parameters ∗par)
- void print_transaction (FILE ∗fp, transaction ∗t)
- void print_block (FILE ∗fp, block ∗b)
- void print_ledger (ledger ∗l)
- void formatted_timestamp (FILE ∗fp)

4.23.1 Function Documentation

4.23.1.1 final_print()

```
void final_print (
             pid_t masterPID,
             user * usersPID,
             node * nodesPID,
             struct parameters * par )
```

Definition at line 40 of file print.c.

4.23.1.2 formatted_timestamp()

```
void formatted_timestamp (
             FILE * fp )
```

Definition at line 157 of file print.c.

4.23.1.3 print_block()

```
void print_block (
             FILE * fp,
             block * b )
```

Definition at line 127 of file print.c.

4.23.1.4 print_kill_signal()

```
void print_kill_signal ( )
```

4.23.1.5 print_ledger()

```
void print_ledger (
             ledger * l )
```

Definition at line 143 of file print.c.

4.23.1.6 print_node_balance()

```
void print_node_balance ( )
```

4.23.1.7 print_num_aborted()

```
void print_num_aborted ( )
```

4.23.1.8 print_num_blocks()

```
void print_num_blocks ( )
```

4.23.1.9 print_parameters()

```
void print_parameters (
             struct parameters * par )
```

Definition at line 52 of file print.c.

4.23.1.10 print_time_to_die()

```
void print_time_to_die ( )
```

Definition at line 7 of file print.c.

4.23.1.11 print_transaction()

```
void print_transaction (
             FILE * fp,
             transaction * t )
```

Definition at line 98 of file print.c.

4.23.1.12 print_transactions_still_in_pool()

```
void print_transactions_still_in_pool ( )
```

4.23.1.13 print_user_balance()

```
void print_user_balance ( )
```

4.23.1.14 print_user_nodes_table()

```
void print_user_nodes_table (
            pid_t mainPID,
            user * userPID,
            node * nodePID,
            struct parameters * par )
```

Definition at line 12 of file print.c.

4.24 print.c

Go to the documentation of this file.

```
00001
00002 #include "include/common.h"
00003 #include "include/print.h"
00004
00005 /* #define HYPHEN "--------" */
00006
00007 void print_time_to_die()
00008 {
00009     printf("\n***********************\n|| The time has come ||\n***********************\n\n");
00010 }
00011
00012 void print_user_nodes_table(pid_t mainPID, user *userPID, node *nodePID, struct parameters *par)
00013 {
00014     int userNum = par->SO_USER_NUM;
00015     int nodesNum = par->SO_NODES_NUM;
00016
00017     printf("\n ------- Master Process PID is %d ----------\n", mainPID);
00018     printf("|                                             |\n");
00019     printf(" - Type ------- PID --------- Status ---------\n");
00020     printf(" ---------------------------------------------\n");
00021     while (userNum--)
00022     {
00023         printf("|  User         %d            %d              |\n", userPID[userNum].pid,
       userPID[userNum].status);
00024     }
00025     printf(" ---------------------------------------------\n");
00026     while (nodesNum--)
00027     {
00028         printf("|  Node         %d            %d              |\n", nodePID[nodesNum].pid,
       nodePID[nodesNum].status);
00029     }
00030     printf(" ---------------------------------------------\n");
00031 }
00032
00033 void print_kill_signal();
00034 void print_user_balance();
00035 void print_node_balance();
00036 void print_num_aborted();
00037 void print_num_blocks();
00038 void print_transactions_still_in_pool();
00039
00040 void final_print(pid_t masterPID, user *usersPID, node *nodesPID, struct parameters *par)
00041 {
00042     print_user_nodes_table(masterPID, usersPID, nodesPID, par);
00043     /*print_kill_signal();
00044     print_user_balance();
00045     print_node_balance();*
00046     print_num_aborted();
00047     /*
00048     print_num_blocks();
00049     print_transactions_still_in_pool();*/
00050 }
```

```

void print_parameters(struct parameters *par)
{
    printf("--------------------------------------------\n----------- Configuration value
 ------------\n");
    printf("SO_USER_NUM->%u\n", par->SO_USER_NUM);
    printf("SO_NODES_NUM->%u\n", par->SO_NODES_NUM);
    printf("SO_BUDGET_INIT->%u\n", par->SO_BUDGET_INIT);
    printf("SO_REWARD->%u\n", par->SO_REWARD);
    printf("SO_MIN_TRANS_GEN_NSEC->%lu\n", par->SO_MIN_TRANS_GEN_NSEC);
    printf("SO_MAX_TRANS_GEN_NSEC->%lu\n", par->SO_MAX_TRANS_GEN_NSEC);
    printf("SO_RETRY->%u\n", par->SO_RETRY);
    printf("SO_TP_SIZE->%u\n", par->SO_TP_SIZE);
    printf("SO_MIN_TRANS_PROC_NSEC->%lu\n", par->SO_MIN_TRANS_PROC_NSEC);
    printf("SO_MAX_TRANS_PROC_NSEC->%lu\n", par->SO_MAX_TRANS_PROC_NSEC);
    printf("SO_SIM_SEC->%u\n", par->SO_SIM_SEC);
    printf("SO_FRIENDS_NUM->%u\n", par->SO_FRIENDS_NUM);
    printf("SO_HOPS->%u\n", par->SO_HOPS);
    printf("--------------------------------------------\n");
}

/*void print_kill_signal(mainPID, userPid /* other process *)
{
    printf("-----PROCESS PID NUM %d KILL----", mainPID);
}
void print_user_balance(int balance)
{
    printf("-----CURRENT BALANCE IS:%d-----", balance);
}

void print_node_balance(int balamce)
{
    printf("----CURRENT NODE BALANCE IS:%d", balance);
} */
/*void print_num_aborted()
{
    printf("\n-- Num of aborted users: %d\n", usersPrematurelyDead);
}
/*
void print_num_blocks()
{
    printf("---TOTAL BLOCK:%d");
}
void print_transactions_still_in_pool()
{
    printf("----TOTAL TRANSACTION STILL IN POLL:%d----");
}*/

void print_transaction(FILE *fp, transaction *t)
{
    char tmp[10];
    switch (t->status)
    {
    case pending:
        strcpy(tmp, "pending");
        break;
    case aborted:
        strcpy(tmp, "aborted");
        break;
    case confirmed:
        strcpy(tmp, "confirmed");
        break;
    case processing:
        strcpy(tmp, "confirmed");
        break;
    }

    fprintf(fp, " -------------------------- \n");
    formatted_timestamp(fp);
    fprintf(fp, "    %s", tmp);
    fprintf(fp, "|  %d --> %d\n", t->sender, t->receiver);
    fprintf(fp, "|  Amount:    %d\n", t->amount);
    fprintf(fp, "|  Reward:    %d\n", t->reward);
    fprintf(fp, "|  Reward:    %d\n", t->reward);
    fprintf(fp, " -------------------------- \n");
}

void print_block(FILE *fp, block *b)
{
    int i;
    block *curr;

    for (curr = b; curr != NULL; curr = (block*)curr->next)
    {
        fprintf(fp, "= %.3d ======================\n", b->blockIndex);
        for (i = 0; i < SO_BLOCK_SIZE; i++)
        {
            print_transaction(fp, &(b->transList[i]));
        }
        fprintf(fp, "============================\n");
    }
}

void print_ledger(ledger *l)
{
    FILE *fp = fopen("ledger.txt", "w");
    if (fp == NULL)
    {
        printf(":print: coudln't open file pointer ledger.txt\n");
    }

    fprintf(fp, "Registry Real Size is %d blocks\n", l->registryCurrSize);
    print_block(fp, l->head);
    fclose(fp);
}

/* print without /n */
void formatted_timestamp(FILE *fp)
{
    printf("Hey");
    /*clock_t tic = clock();
    clock_t start = clock();
    clock_t stop = clock();

    time_t rawtime;
    time_t now;
    struct tm *info;
    struct tm *today;
    double elapsed;
    char buf[128];

    time(&now);
    today = localtime(&now);
    strftime(buf, 128, "%Y/%m/%d", today);
    printf("%s\n", buf);

    elapsed = (double)(stop - start) * 1000.0 / CLOCKS_PER_SEC; /* time ./a.out*/
}
```

4.25 src/users.c File Reference

```
#include "include/common.h"
#include "include/users.h"
```

Include dependency graph for users.c:

Macros

- #define REWARD(amount, reward) (ceil(((reward ∗ (amount)) / 100.0)))

Functions

- int get_pid_userIndex (int PID_toSearch)
- pid_t get_random_userPID ()
- pid_t get_random_nodePID ()
- void update_status (int statusToSet)
- void attach_ipc_objects (char ∗∗argv)
- void queue_to_pid (pid_t nodePID)
- void transaction_init (pid_t userPID, int amount, int reward)
- void signal_handlers_init (struct sigaction ∗saUSR1, struct sigaction ∗saINT)
- int send_transaction ()
- void user_transactions_handle (int signum)
- void user_interrupt_handle (int signum)
- int main (int argc, char ∗argv[ ])

Variables

- struct parameters ∗ par
- user ∗ usersPID
- node ∗ nodesPID
- ledger ∗ mainLedger
- int semID
- int queueID
- int currBalance
- pid_t myPID
- int outGoingTransactions
- transaction currTrans

4.25.1 Macro Definition Documentation

4.25.1.1 REWARD

```
#define REWARD(
              amount,
              reward ) (ceil(((reward * (amount)) / 100.0)))
```

Definition at line 4 of file users.c.

4.25.2 Function Documentation

4.25.2.1 attach_ipc_objects()

```
void attach_ipc_objects (
             char ** argv )
```

Definition at line 120 of file users.c.

4.25.2.2 get_pid_userIndex()

```
int get_pid_userIndex (
             int PID_toSearch )
```

Definition at line 51 of file users.c.

4.25.2.3 get_random_nodePID()

```
pid_t get_random_nodePID ( )
```

Definition at line 83 of file users.c.

4.25.2.4 get_random_userPID()

```
pid_t get_random_userPID ( )
```

Definition at line 65 of file users.c.

4.25.2.5 main()

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

Definition at line 233 of file users.c.

4.25.2.6 queue_to_pid()

```
void queue_to_pid (
             pid_t nodePID )
```

Definition at line 138 of file users.c.

4.25.2.7 send_transaction()

```
int send_transaction ( )
```

Definition at line 176 of file users.c.

4.25.2.8 signal_handlers_init()

```
void signal_handlers_init (
             struct sigaction * saUSR1,
             struct sigaction * saINT )
```

Definition at line 160 of file users.c.

4.25.2.9 transaction_init()

```
void transaction_init (
             pid_t userPID,
             int amount,
             int reward )
```

Definition at line 146 of file users.c.

4.25.2.10 update_status()

```
void update_status (
             int statusToSet )
```

Definition at line 101 of file users.c.

4.25.2.11 user_interrupt_handle()

```
void user_interrupt_handle (
             int signum )
```

Definition at line 227 of file users.c.

4.25.2.12 user_transactions_handle()

```
void user_transactions_handle (
             int signum )
```

Definition at line 217 of file users.c.

4.25.3 Variable Documentation

4.25.3.1 currBalance

```
int currBalance
```

Definition at line 39 of file users.c.

4.25.3.2 currTrans

```
transaction currTrans
```

Definition at line 42 of file users.c.

4.25.3.3 mainLedger

```
ledger* mainLedger
```

Definition at line 34 of file users.c.

4.25.3.4 myPID

```
pid_t myPID
```

Definition at line 40 of file users.c.

4.25.3.5 nodesPID

```
node* nodesPID
```

Definition at line 33 of file users.c.

4.25.3.6 outGoingTransactions

```
int outGoingTransactions
```

Definition at line 41 of file users.c.

4.25.3.7 par

```
struct parameters* par
```

Definition at line 31 of file users.c.

4.25.3.8 queueID

```
int queueID
```

Definition at line 37 of file users.c.

4.25.3.9 semID

```
int semID
```

Definition at line 36 of file users.c.

4.25.3.10 usersPID

```
user* usersPID
```

Definition at line 32 of file users.c.

4.26 users.c

Go to the documentation of this file.

```
00001 #include "include/common.h"
00002 #include "include/users.h"
00003
00004 #define REWARD(amount, reward) (ceil(((reward * (amount)) / 100.0)))
00005 /*
00006  * NON active wait, the time is equivalent to the
00007  * verification algorithms that happen in "real" blockchains
00008  */
00009
00010 /*
00011  * Need to implement a way to send s transaction
00012  * signal, we can utilize a user defined signal
00013  * handler.
00014  * We also need to account for the signal SIGINT (CTRL-C).
00015  * Maybe we can implement some sort of graphic way to visualize
00016  * child processes (nodes and user) so that we can choose
00017  * the PID on which to send the signal to.
00018  * -- user-defined signal handlers are inherited by the child processes --
00019  * so it's better to handle them in the master program
00020  */
00021
00022 /* void wait_for_incoming_transaction() */
00023
00024 /*
00025  ======================
00026  || GLOBAL VARIABLES ||
00027  ======================
00028  */
00029
00030 /* parameters of simulation */
00031 struct parameters *par;
00032 user *usersPID;
00033 node *nodesPID;
00034 ledger *mainLedger;
00035
00036 int semID;
00037 int queueID;
00038
00039 int currBalance;
00040 pid_t myPID;
```

```c
int outGoingTransactions; /* accumulate amount of transactions sent but yet to be received */
transaction currTrans;

/*
 ======================
 ||    FUNCTIONS     ||
 ======================
 */

/* returns index of where user with PID_toSearch is located in usersPID[] */
int get_pid_userIndex(int PID_toSearch)
{
    int i;

    for (i = 0; i < par->SO_USER_NUM; i++)
    {
        if (usersPID[i].pid == myPID)
            return i;
    }

    return -1;
}

/* returns a random PID of a non-dead user from usersPID[] */
pid_t get_random_userPID()
{
    int index;
    pid_t val = 0;

    do
    {
        index = RAND(0, par->SO_USER_NUM - 1);
        TRACE((":user: %d index is %d\n", myPID, index))
        TRACE((":users: %d usersPID[%d]\n", myPID, index));
        if (usersPID[index].status != dead)
            val = usersPID[index].pid;
    } while (!val);

    return val;
}

/* returns a random PID of an available node from nodesPID[] */
pid_t get_random_nodePID()
{
    int index;
    pid_t val = 0;

    do
    {
        index = RAND(0, par->SO_NODES_NUM - 1);
        TRACE((":user: %d index is %d\n", myPID, index))
        TRACE((":users: %d nodesPID[%d]\n", myPID, index));
        if (nodesPID[index].status == available)
            val = nodesPID[index].pid;
    } while (!val);

    return val;
}

/* safely updates status of user to statusToSet: 0 alive, 1 broke, 2 dead */
void update_status(int statusToSet)
{
    int i = get_pid_userIndex(myPID);
    if (i == -1)
    {
        TRACE((":users: %d failed to find myself in usersPID[]", myPID));
    }

    sem_reserve(semID, 1);
    usersPID[i].status = statusToSet;
    if (statusToSet == 2)
    {
        /*usersPrematurelyDead++;*/
        TRACE((":users: dead increased\n"));
    }
    sem_release(semID, 1);
}

/* attaches ipc objects based on IDs passed via arguments */
void attach_ipc_objects(char **argv)
{
    par = shmat(PARAMETERS_ARGV, NULL, 0);
    TRACE((":users %d par->SO_RETRY %d\n", myPID, par->SO_RETRY))
    TEST_ERROR
    usersPID = shmat(USERS_PID_ARGV, NULL, 0);
    TRACE((":users: %d usersPID[0] = %d, usersPID[3] = %d\n", myPID, usersPID[0], usersPID[3]))
    TEST_ERROR
    nodesPID = shmat(NODES_PID_ARGV, NULL, 0);
    TRACE((":users: %d nodesPID[0] = %d, nodesPID[3] = %d\n", myPID, nodesPID[0], nodesPID[3]))
    TEST_ERROR
    mainLedger = shmat(LEDGER_ARGV, NULL, 0);
    TEST_ERROR
    semID = SEM_ID_ARGV;
    TRACE((":users: %d semID is %d\n", myPID, semID));
}

/* use nodePID as key for msgget and check for errors */
void queue_to_pid(pid_t nodePID)
{
    queueID = msgget(nodePID, IPC_CREAT | 0600);
    TEST_ERROR
    TRACE((":users: %d -> %d queueID %d\n", myPID, nodePID, queueID))
}

/* initializes transaction values and sets it to pending */
void transaction_init(pid_t userPID, int amount, int reward)
{
    struct timespec exactTime;

    currTrans.sender = myPID;
    currTrans.receiver = userPID;
    currTrans.amount = amount;
    currTrans.reward = reward;
    currTrans.status = pending;
    clock_gettime(CLOCK_REALTIME, &exactTime);
    currTrans.timestamp = exactTime;
}

/* initializes signal handlers for SIGINT and SIGUSR1 */
void signal_handlers_init(struct sigaction *saUSR1, struct sigaction *saINT)
{
    /* -- SIGNAL HANDLERS --
     * first set all bytes of sigation to 0
     * then initialize sa.handler to a pointer to
     * the function user_transaction/interrupt_handle
     * then set the handler to handle SIUSR1/SIGINT signals
     * ((struct sigaction *oldact) = NULL)
     */
    saUSR1->sa_handler = user_transactions_handle;
    saINT->sa_handler = user_interrupt_handle;
    sigaction(SIGUSR1, saUSR1, NULL);
    sigaction(SIGINT, saINT, NULL);
}

/* send transaction currTrans to user userPID via node nodePID */
int send_transaction()
{
    msgsnd(queueID, &currTrans, sizeof(transaction), 0);
    TEST_ERROR
    currBalance -= (currTrans.amount + currTrans.reward);
    outGoingTransactions += (currTrans.amount + currTrans.reward);
    switch (errno)
    {
    case EACCES:
        printf(":users %d no write permission on queue\n", myPID);
        break;
    case EAGAIN:
        printf(":users: %d queue full\n", myPID); /* keep if we decide to use IPC_NOWAIT */
        break;
    case EFAULT:
        printf(":users: %d address pointed by msgp inaccessible\n", myPID);
        break;
    case EIDRM:
        printf(":users: %d message queue removed\n", myPID);
        break;
    case EINTR:
        TRACE((":users: %d signal caught when waiting for queue to free\n", myPID));
        break;
    case EINVAL:
        printf(":users: %d invalid  msqid  value,  or nonpositive mtype value, or invalid msgsz
 value\n", myPID);
        break;
    case ENOMEM:
        printf(":users: %d system out of memory\n", myPID); /* should basically never happen I hope */
        break;
    default:
        TRACE(("Transaction sent\n"))
        return SUCCESS;
    }
    currTrans.status = aborted;
    currBalance += (currTrans.amount + currTrans.reward);
    outGoingTransactions -= (currTrans.amount + currTrans.reward);
    /* we can then track this type of aborted transactions but rn there's no need to */
    return ERROR;
```

```
}

/* SIGUSR1 handler, sends a transaction */
void user_transactions_handle(int signum)
{
    write(1, "::User:: SIGUSR1 received\n", 27);
    if (currBalance > 2)
        send_transaction(); /* we're calling a printf which is not thread safe, need to fix somehow*/
    else
        write(1, "::User:: sorry balance too low\n", 32);
}

/* CTRL-C handler */
void user_interrupt_handle(int signum)
{
    write(1, "::User:: SIGINT received\n", 26);
    exit(0);
}

int main(int argc, char *argv[])
{
    int amount, reward, retry;
    pid_t userPID, nodePID;

    struct timespec randSleepTime;
    struct timespec sleepTimeRemaining;

    struct sembuf sops;
    struct message transMsg;

    struct sigaction saUSR1;
    struct sigaction saINT;
    bzero(&saUSR1, sizeof(saUSR1));
    bzero(&saINT, sizeof(saINT));

    myPID = getpid(); /* set myPID value */
    TRACE((":users: %d USERS_PID_ARGV %d\n", myPID, USERS_PID_ARGV))
    TRACE((":users: %d NODES_PID_ARGV %d\n", myPID, NODES_PID_ARGV))
    TRACE((":users: %d PARAMETERS_ARGV %d\n", myPID, PARAMETERS_ARGV))
    TRACE((":users: %d LEDGER_ARGV %d\n", myPID, LEDGER_ARGV))
    TRACE((":users: %d SEM_ID_PID_ARGV %d\n", myPID, SEM_ID_ARGV))

    if (argc == 0)
    {
        printf(":users: %d, no arguments passed, can't continue like this any more :C\n", myPID);
        return ERROR;
    }

    srand(time(NULL)); /* initialize rand function */

    attach_ipc_objects(argv);
    signal_handlers_init(&saUSR1, &saINT);
    transMsg.mtype = atol("transaction");

    retry = par->SO_RETRY;
    while (1)
    {
        SLEEP_TIME_SET
        /*
         * save the time unslept when interrupted by SIGUSR1
         * so that we can't force transactions at a much greater speed
         * better to save it into a separate struct because clock_nanosleep
         * will not update it if the sleep is not interrupted
         */
        bzero(&sleepTimeRemaining, sizeof(sleepTimeRemaining));

        currBalance = 100 /*balance(myPID)*/;
        if (currBalance >= 2)
        {
            userPID = get_random_userPID();
            nodePID = get_random_nodePID();

            amount = RAND(2, currBalance);
            reward = REWARD(amount, par->SO_REWARD);
            amount -= reward;

            queue_to_pid(nodePID);
            transaction_init(userPID, amount, reward);
            if (send_transaction() == 0)
                retry = par->SO_RETRY;
            else
                retry--;

            if (retry == 0)
            {
                update_status(2);
                return MAX_RETRY;
            }
            SLEEP
        }
        else
        {
            printf(":users: %d went broke :/\n", myPID);
            update_status(1);

            /*wait_for_incoming_transaction(); ///////// */
        }
    }
}
```

4.27 src/utils/debug.c File Reference

`#include "debug.h"`

Include dependency graph for debug.c:

Functions

- void dbg_printf (const char *fmt,...)

4.27.1 Function Documentation

4.27.1.1 dbg_printf()

```
void dbg_printf (
            const char * fmt,
             ... )
```

Definition at line 3 of file debug.c.

4.28 debug.c

Go to the documentation of this file.

```
00001 #include "debug.h"
00002
00003 void dbg_printf(const char *fmt, ...)
00004 {
00005     va_list args;
00006     va_start(args, fmt);
00007     vfprintf(stderr, fmt, args);
00008     va_end(args);
00009 }
```

4.29 src/utils/debug.h File Reference

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

Include dependency graph for debug.h: This graph shows which files directly or indirectly include this file:

Macros

- #define TRACE(x)

Functions

- void dbg_printf (const char ∗fmt,...)

4.29.1 Macro Definition Documentation

4.29.1.1 TRACE

```
#define TRACE(
               x )
```

Definition at line 12 of file debug.h.

4.29.2 Function Documentation

4.29.2.1 dbg_printf()

```
void dbg_printf (
              const char * fmt,
               ...  )
```

Definition at line 3 of file debug.c.

4.30 debug.h

Go to the documentation of this file.

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_DEBUG_H
00002 #define SIMULAZIONE_TRANSAZIONI_DEBUG_H
00003
00004 #include <stdarg.h>
00005 #include <stdio.h>
00006 /* kudos to
       https://stackoverflow.com/questions/1644868/define-macro-for-debug-printing-in-c/1644898#1644898?newreg=f3d84f0e4a0846
       */
00007
00008 void dbg_printf(const char *fmt, ...);
00009 #ifdef DEBUG
00010 #define TRACE(x) dbg_printf x;
00011 #else
00012 #define TRACE(x)
00013 #endif
00014
00015 #endif /* SIMULAZIONE_TRANSAZIONI_DEBUG_H */
```

4.31 src/utils/lists.c File Reference

4.32 lists.c

Go to the documentation of this file.

4.33 src/utils/lists.h File Reference

This graph shows which files directly or indirectly include this file:

4.34 lists.h

Go to the documentation of this file.

4.35 src/utils/sem.c File Reference

```
#include "sem.h"
```
Include dependency graph for sem.c:

Macros

- #define TEST_ERROR

Functions

- int sem_set_val (int sem_id, int sem_num, int sem_val)
- int sem_reserve (int sem_id, int sem_num)
- int sem_release (int sem_id, int sem_num)
- int sem_getall (char ∗my_string, int sem_id)

Variables

- int errno

4.35.1 Macro Definition Documentation

4.35.1.1 TEST_ERROR

```
#define TEST_ERROR
```

Value:

```
    if (errno)                                      \
    {                                               \
        fprintf(stderr,                             \
                "%s:%d: PID=%5d: Error %d (%s)\n", \
                __FILE__,                           \
                __LINE__,                           \
                getpid(),                           \
                errno,                              \
                strerror(errno));                   \
    }
```

Definition at line 4 of file sem.c.

4.35.2 Function Documentation

4.35.2.1 sem_getall()

```
int sem_getall (
              char * my_string,
              int sem_id )
```

Definition at line 49 of file sem.c.

4.35.2.2 sem_release()

```
int sem_release (
              int sem_id,
              int sem_num )
```

Definition at line 37 of file sem.c.

4.35.2.3 sem_reserve()

```
int sem_reserve (
              int sem_id,
              int sem_num )
```

Definition at line 26 of file sem.c.

4.35.2.4 sem_set_val()

```
int sem_set_val (
            int sem_id,
            int sem_num,
            int sem_val )
```

Definition at line 20 of file sem.c.

4.35.3 Variable Documentation

4.35.3.1 errno

```
int errno
```

Definition at line 17 of file sem.c.

4.36 sem.c

Go to the documentation of this file.

```
00001 #include "sem.h"
00002 
00003 #ifndef TEST_ERROR
00004 #define TEST_ERROR                                      \
00005     if (errno)                                          \
00006     {                                                   \
00007         fprintf(stderr,                                 \
00008                 "%s:%d: PID=%5d: Error %d (%s)\n",      \
00009                 __FILE__,                               \
00010                 __LINE__,                               \
00011                 getpid(),                               \
00012                 errno,                                  \
00013                 strerror(errno));                       \
00014     }
00015 #endif
00016 
00017 int errno;
00018 
00019 /* Set a semaphore to a user defined value */
00020 int sem_set_val(int sem_id, int sem_num, int sem_val)
00021 {
00022     return semctl(sem_id, sem_num, SETVAL, sem_val);
00023 }
00024 
00025 /* Try to access the resource */
00026 int sem_reserve(int sem_id, int sem_num)
00027 {
00028     struct sembuf sops;
00029 
00030     sops.sem_num = sem_num;
00031     sops.sem_op = -1;
00032     sops.sem_flg = 0;
00033     return semop(sem_id, &sops, 1);
00034 }
00035 
00036 /* Release the resource */
00037 int sem_release(int sem_id, int sem_num)
00038 {
00039     struct sembuf sops;
00040 
00041     sops.sem_num = sem_num;
00042     sops.sem_op = 1;
00043     sops.sem_flg = 0;
00044 
```

```
    return semop(sem_id, &sops, 1);
}

/* Print all semaphore values to a string */
int sem_getall(char *my_string, int sem_id)
{
    union semun arg; /* man semctl per vedere def della union  */
    unsigned short *sem_vals, i;
    unsigned long num_sem;
    char cur_str[10];
    struct semid_ds my_ds;

    /* Get the number of semaphores */
    arg.buf = &my_ds;
    semctl(sem_id, 0, IPC_STAT, arg);
    TEST_ERROR;
    num_sem = arg.buf->sem_nsems;

    /* Get the values of all semaphores */
    sem_vals = malloc(sizeof(*sem_vals) * num_sem);
    arg.array = sem_vals;
    semctl(sem_id, 0, GETALL, arg);

    /* Initialize the string. MUST be allocated by the caller */
    my_string[0] = 0;
    for (i = 0; i < num_sem; i++)
    {
        sprintf(cur_str, "%d ", sem_vals[i]);
        strcat(my_string, cur_str);
    }
}
```

4.37 src/utils/sem.h File Reference

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
```

Include dependency graph for sem.h: This graph shows which files directly or indirectly include this file:

Data Structures

- union semun

Macros

- #define LOCK
- #define UNLOCK

Functions

- int sem_set_val (int sem_id, int sem_num, int sem_val)
- int sem_reserve (int sem_id, int sem_num)
- int sem_release (int sem_id, int sem_num)
- int sem_getall (char ∗my_string, int sem_id)

4.37.1 Macro Definition Documentation

4.37.1.1 LOCK

```
#define LOCK
```

Value:
```
    sops.sem_num = 1;            \
    sops.sem_op = -1;            \
    sops.sem_flg = 0;            \
    semop(semID, &sops, 1);
```

Definition at line 14 of file sem.h.

4.37.1.2 UNLOCK

```
#define UNLOCK
```

Value:
```
    sops.sem_num = 1;            \
    sops.sem_op = 1;             \
    sops.sem_flg = 0;            \
    semop(semID, &sops, 1);
```

Definition at line 19 of file sem.h.

4.37.2 Function Documentation

4.37.2.1 sem_getall()

```
int sem_getall (
             char * my_string,
             int sem_id )
```

Definition at line 49 of file sem.c.

4.37.2.2 sem_release()

```
int sem_release (
             int sem_id,
             int sem_num )
```

Definition at line 37 of file sem.c.

4.37.2.3 sem_reserve()

```
int sem_reserve (
             int sem_id,
             int sem_num )
```

Definition at line 26 of file sem.c.

4.37.2.4 sem_set_val()

```
int sem_set_val (
             int sem_id,
             int sem_num,
             int sem_val )
```

Definition at line 20 of file sem.c.

4.38 sem.h

Go to the documentation of this file.

```
#ifndef SIMULAZIONE_TRANSAZIONI_SEM_H
#define SIMULAZIONE_TRANSAZIONI_SEM_H

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

/* from prof. Schifanella examples */
#define LOCK                         \
    sops.sem_num = 1;                \
    sops.sem_op = -1;                \
    sops.sem_flg = 0;                \
    semop(semID, &sops, 1);
#define UNLOCK                       \
    sops.sem_num = 1;                \
    sops.sem_op = 1;                 \
    sops.sem_flg = 0;                \
    semop(semID, &sops, 1);

/* from prof. Bini examples */

/*
 * The following union must be defined as required by the semctl man
 * page
 */
union semun {
    int              val;    /* Value for SETVAL */
    struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
    unsigned short  *array;  /* Array for GETALL, SETALL */
    struct seminfo  *__buf;  /* Buffer for IPC_INFO
                            (Linux-specific) */
};


/*
 * Set a semaphore to a user defined value
 * INPUT:
 * - sem_id: the ID of the semaphore IPC object
 * - sem_num: the position of the semaphore in the array
 * - sem_val: the initialization value of the semaphore
 * RESULT:
 * - the selected semaphore is initialized to the given value
 * - the returned value is the same as the invoked semctl
 */
int sem_set_val(int sem_id, int sem_num, int sem_val);

/*
 * Try to access the resource
 * INPUT:
 * - sem_id: the ID of the semaphore IPC object
 * - sem_num: the position of the semaphore in the array
 * RESULT
 * - if the resource is available (semaphore value > 0), the semaphore
 *   is decremented by one
 * - if the resource is not available (semaphore value == 0), the
 *   process is blocked until the resource becomes available again
 * - the returned value is the same as the invoked semop
 */
int sem_reserve(int sem_id, int sem_num);

/*
 * Release the resource
 * INPUT:
 * - sem_id: the ID of the semaphore IPC object
 * - sem_num: the position of the semaphore in the array
 * RESULT:
 * - the semaphore value is incremented by one. This may unblock some
 *   process
 * - the returned value is the same as the invoked semop
 */
int sem_release(int sem_id, int sem_num);

/*
 * Print all semaphore values to a string. my_string MUST be
 * previously allocated
 */
int sem_getall(char * my_string, int sem_id);


#endif /* SIMULAZIONE_TRANSAZIONI_SEM_H */
```

Index