

Simulazione Transazioni

Generated by Doxygen 1.9.3

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 block_t Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 blockIndex	5
3.1.2.2 next	6
3.1.2.3 prev	6
3.1.2.4 transList	6
3.2 ledger_t Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Field Documentation	6
3.2.2.1 head	7
3.2.2.2 registryCurrSize	7
3.3 message Struct Reference	7
3.3.1 Detailed Description	7
3.3.2 Field Documentation	7
3.3.2.1 mtype	7
3.3.2.2 userTrans	8
3.4 node_t Struct Reference	8
3.4.1 Detailed Description	8
3.4.2 Member Enumeration Documentation	8
3.4.2.1 anonymous enum	8
3.4.3 Field Documentation	9
3.4.3.1 pid	9
3.4.3.2	9
3.5 parameters Struct Reference	9
3.5.1 Detailed Description	9
3.5.2 Field Documentation	10
3.5.2.1 SO_BUDGET_INIT	10
3.5.2.2 SO_FRIENDS_NUM	10
3.5.2.3 SO_HOPS	10
3.5.2.4 SO_MAX_TRANS_GEN_NSEC	10
3.5.2.5 SO_MAX_TRANS_PROC_NSEC	10
3.5.2.6 SO_MIN_TRANS_GEN_NSEC	11
3.5.2.7 SO_MIN_TRANS_PROC_NSEC	11
3.5.2.8 SO_NODES_NUM	11
3.5.2.9 SO_RETRY	11

3.5.2.10 SO_REWARD	11
3.5.2.11 SO_SIM_SEC	11
3.5.2.12 SO_TP_SIZE	12
3.5.2.13 SO_USER_NUM	12
3.6 semun Union Reference	12
3.6.1 Detailed Description	12
3.6.2 Field Documentation	12
3.6.2.1 __buf	12
3.6.2.2 array	13
3.6.2.3 buf	13
3.6.2.4 val	13
3.7 transaction_t Struct Reference	13
3.7.1 Detailed Description	13
3.7.2 Member Enumeration Documentation	13
3.7.2.1 anonymous enum	13
3.7.3 Field Documentation	14
3.7.3.1 amount	14
3.7.3.2 receiver	14
3.7.3.3 reward	14
3.7.3.4 sender	14
3.7.3.5	15
3.7.3.6 timestamp	15
3.8 user_t Struct Reference	15
3.8.1 Detailed Description	15
3.8.2 Member Enumeration Documentation	15
3.8.2.1 anonymous enum	15
3.8.3 Field Documentation	16
3.8.3.1 pid	16
3.8.3.2	16
4 File Documentation	17
4.1 src/common.c File Reference	17
4.2 common.c	17
4.3 src/include/balance.h File Reference	18
4.3.1 Function Documentation	18
4.3.1.1 balance()	18
4.4 balance.h	18
4.5 src/include/common.h File Reference	18
4.5.1 Macro Definition Documentation	20
4.5.1.1 _GNU_SOURCE	20
4.5.1.2 ARRAY_SIZE	20
4.5.1.3 ERROR	20

4.5.1.4 EVERYONE_BROKE	20
4.5.1.5 IPC_ERROR	20
4.5.1.6 LEDGER_ARGV	21
4.5.1.7 M_QUEUE_KEY	21
4.5.1.8 MAX_RETRY	21
4.5.1.9 NODES_PID_ARGV	21
4.5.1.10 NULL	21
4.5.1.11 PARAMETERS_ARGV	21
4.5.1.12 RAND	22
4.5.1.13 SELF	22
4.5.1.14 SEM_ID_ARGV	22
4.5.1.15 SEM_MASTER	22
4.5.1.16 SHM_LEDGER	22
4.5.1.17 SHM_NODES_ARRAY	22
4.5.1.18 SHM_PARAMETERS	23
4.5.1.19 SHM_USERS_ARRAY	23
4.5.1.20 SO_BLOCK_SIZE	23
4.5.1.21 SO_REGISTRY_SIZE	23
4.5.1.22 SUCCESS	23
4.5.1.23 TEST_ERROR	23
4.5.1.24 USERS_PID_ARGV	24
4.5.1.25 WENT_BROKE	24
4.5.2 Typedef Documentation	24
4.5.2.1 block	24
4.5.2.2 ledger	24
4.5.2.3 node	24
4.5.2.4 transaction	24
4.5.2.5 user	24
4.5.3 Function Documentation	25
4.5.3.1 add_block()	25
4.5.3.2 add_block_to_ledger()	25
4.5.3.3 add_transaction_to_block()	25
4.5.3.4 find_transaction()	25
4.5.3.5 ledger_init()	25
4.5.3.6 new_block()	26
4.5.3.7 search_receiver()	26
4.5.3.8 search_sender()	26
4.5.3.9 search_timestamp()	26
4.5.4 Variable Documentation	26
4.5.4.1 errno	26
4.6 common.h	27
4.7 src/include/master.h File Reference	29

4.7.1 Function Documentation	29
4.7.1.1 make_arguments()	29
4.7.1.2 make_ipc_array()	29
4.7.1.3 master_interrupt_handle()	29
4.7.1.4 semaphores_init()	30
4.7.1.5 shared_memory_objects_init()	30
4.7.1.6 spawn_node()	30
4.7.1.7 spawn_user()	30
4.8 master.h	30
4.9 src/include/nodes.h File Reference	31
4.9.1 Variable Documentation	31
4.9.1.1 randSleepTime	31
4.9.1.2 sleepTimeRemaining	32
4.10 nodes.h	32
4.11 src/include/parser.h File Reference	32
4.11.1 Macro Definition Documentation	32
4.11.1.1 CONF_ERROR	32
4.11.1.2 CONF_FILE	32
4.11.1.3 NUM_PARAMETERS	33
4.11.2 Function Documentation	33
4.11.2.1 assign_defaults()	33
4.11.2.2 parse_parameters()	33
4.12 parser.h	33
4.13 src/include/print.h File Reference	33
4.13.1 Function Documentation	34
4.13.1.1 final_print()	34
4.13.1.2 formatted_timestamp()	34
4.13.1.3 print_block()	34
4.13.1.4 print_kill_signal()	35
4.13.1.5 print_ledger()	35
4.13.1.6 print_node_balance()	35
4.13.1.7 print_num_aborted()	35
4.13.1.8 print_num_blocks()	35
4.13.1.9 print_parameters()	35
4.13.1.10 print_time_to_die()	35
4.13.1.11 print_transaction()	36
4.13.1.12 print_transactions_still_in_pool()	36
4.13.1.13 print_user_balance()	36
4.13.1.14 print_user_nodes_table()	36
4.14 print.h	36
4.15 src/include/users.h File Reference	37
4.15.1 Macro Definition Documentation	37

4.15.1.1 SLEEP	37
4.15.1.2 SLEEP_TIME_SET	37
4.15.2 Function Documentation	38
4.15.2.1 get_pid_userIndex()	38
4.15.2.2 get_random_nodePID()	38
4.15.2.3 get_random_userPID()	38
4.15.2.4 queue_to_pid()	38
4.15.2.5 send_transaction()	38
4.15.2.6 transaction_init()	39
4.15.2.7 user_interrupt_handle()	39
4.15.2.8 user_transactions_handle()	39
4.16 users.h	39
4.17 src/master.c File Reference	40
4.17.1 Macro Definition Documentation	40
4.17.1.1 IPC_NUM	40
4.17.1.2 NODE_NAME	41
4.17.1.3 SEM_NUM	41
4.17.1.4 SHM_NUM	41
4.17.1.5 USER_NAME	41
4.17.2 Function Documentation	41
4.17.2.1 main()	41
4.17.2.2 make_arguments()	42
4.17.2.3 make_ipc_array()	42
4.17.2.4 master_interrupt_handle()	42
4.17.2.5 semaphores_init()	42
4.17.2.6 shared_memory_objects_init()	42
4.17.2.7 spawn_node()	42
4.17.2.8 spawn_user()	43
4.17.3 Variable Documentation	43
4.17.3.1 mainLedger	43
4.17.3.2 nodesPID	43
4.17.3.3 par	43
4.17.3.4 semID	43
4.17.3.5 usersPID	43
4.18 master.c	44
4.19 src/nodes.c File Reference	47
4.19.1 Function Documentation	47
4.19.1.1 main()	47
4.19.2 Variable Documentation	48
4.19.2.1 mainLedger	48
4.19.2.2 myPID	48
4.19.2.3 nodesPID	48

4.19.2.4 par	48
4.19.2.5 queueID	48
4.19.2.6 semID	49
4.19.2.7 usersPID	49
4.20 nodes.c	49
4.21 src/parser.c File Reference	50
4.21.1 Function Documentation	50
4.21.1.1 assign_defaults()	50
4.21.1.2 parse_parameters()	50
4.22 parser.c	51
4.23 src/print.c File Reference	53
4.23.1 Function Documentation	53
4.23.1.1 final_print()	54
4.23.1.2 formatted_timestamp()	54
4.23.1.3 print_block()	54
4.23.1.4 print_kill_signal()	54
4.23.1.5 print_ledger()	54
4.23.1.6 print_node_balance()	54
4.23.1.7 print_num_aborted()	55
4.23.1.8 print_num_blocks()	55
4.23.1.9 print_parameters()	55
4.23.1.10 print_time_to_die()	55
4.23.1.11 print_transaction()	55
4.23.1.12 print_transactions_still_in_pool()	55
4.23.1.13 print_user_balance()	56
4.23.1.14 print_user_nodes_table()	56
4.24 print.c	56
4.25 src/users.c File Reference	58
4.25.1 Macro Definition Documentation	59
4.25.1.1 REWARD	59
4.25.2 Function Documentation	59
4.25.2.1 attach_ipc_objects()	59
4.25.2.2 get_pid_userIndex()	59
4.25.2.3 get_random_nodePID()	60
4.25.2.4 get_random_userPID()	60
4.25.2.5 main()	60
4.25.2.6 queue_to_pid()	60
4.25.2.7 send_transaction()	60
4.25.2.8 signal_handlers_init()	60
4.25.2.9 transaction_init()	61
4.25.2.10 update_status()	61
4.25.2.11 user_interrupt_handle()	61

4.25.2.12 user_transactions_handle()	61
4.25.3 Variable Documentation	61
4.25.3.1 currBalance	61
4.25.3.2 currTrans	62
4.25.3.3 mainLedger	62
4.25.3.4 myPID	62
4.25.3.5 nodesPID	62
4.25.3.6 outGoingTransactions	62
4.25.3.7 par	62
4.25.3.8 queueID	63
4.25.3.9 semID	63
4.25.3.10 usersPID	63
4.26 users.c	63
4.27 src/utils/debug.c File Reference	67
4.27.1 Function Documentation	67
4.27.1.1 dbg_printf()	67
4.28 debug.c	67
4.29 src/utils/debug.h File Reference	67
4.29.1 Macro Definition Documentation	68
4.29.1.1 TRACE	68
4.29.2 Function Documentation	68
4.29.2.1 dbg_printf()	68
4.30 debug.h	68
4.31 src/utils/lists.c File Reference	69
4.32 lists.c	69
4.33 src/utils/lists.h File Reference	69
4.34 lists.h	69
4.35 src/utils/sem.c File Reference	69
4.35.1 Macro Definition Documentation	69
4.35.1.1 TEST_ERROR	70
4.35.2 Function Documentation	70
4.35.2.1 sem_getall()	70
4.35.2.2 sem_release()	70
4.35.2.3 sem_reserve()	70
4.35.2.4 sem_set_val()	71
4.35.3 Variable Documentation	71
4.35.3.1 errno	71
4.36 sem.c	71
4.37 src/utils/sem.h File Reference	72
4.37.1 Macro Definition Documentation	73
4.37.1.1 LOCK	73
4.37.1.2 UNLOCK	73

4.37.2 Function Documentation	73
4.37.2.1 sem_getall()	73
4.37.2.2 sem_release()	73
4.37.2.3 sem_reserve()	74
4.37.2.4 sem_set_val()	74
4.38 sem.h	74
Index	77

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

block_t	5
ledger_t	6
message	7
node_t	8
parameters	9
semun	12
transaction_t	13
user_t	15

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/ common.c	17
src/ master.c	40
src/ nodes.c	47
src/ parser.c	50
src/ print.c	53
src/ users.c	58
src/include/ balance.h	18
src/include/ common.h	18
src/include/ master.h	29
src/include/ nodes.h	31
src/include/ parser.h	32
src/include/ print.h	33
src/include/ users.h	37
src/utls/ debug.c	67
src/utls/ debug.h	67
src/utls/ lists.c	69
src/utls/ lists.h	69
src/utls/ sem.c	69
src/utls/ sem.h	72

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/utls/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.](#)

```
00001 #include "include/common.h"
00002
00003 ledger *ledger_init()
00004 {
00005     ledger *newLedger;
00006     int shmID; /* ID of "ledger" shared memory segment */
00007
00008     /* -- LEDGER INITIALIZATION --
00009      * save the ID of our new (IPC_PRIVATE) shared memory segment of size -ledger-
00010      * smctl will deallocate the shared memory segment only when every process detaches it
00011      * tells OS that ledger of type ledger is our shared memory of shmID
00012      */
00013     shmID = shmget(IPC_PRIVATE, sizeof(newLedger), 0600);
00014     shmctl(shmID, IPC_RMID, NULL);
00015
00016     newLedger->head = new_block();
00017     newLedger->registryCurrSize = 1;
00018
00019     newLedger = (ledger *)shmat(shmID, NULL, 0);
00020
00021     return newLedger;
00022 }
00023
00024 block *new_block()
00025 {
00026     block *newBlock = malloc(sizeof(block));
00027     transaction reward;
00028     struct timespec timestamp;
00029
00030     /* memset(newBlock->transList, 0, SO_BLOCK_SIZE); */
00031     clock_gettime(CLOCK_REALTIME, &timestamp);
00032
00033     reward.timestamp = timestamp;
00034     reward.sender = SELF;
00035     reward.receiver = getpid();
00036     reward.amount = 0;
00037     reward.reward = 0;
00038
00039     newBlock->transList[0] = reward;
00040     newBlock->blockIndex = 0;
00041     newBlock->next = NULL;
00042
00043     return newBlock;
00044 }
00045
00046 void add_transaction_to_block(block *block, transaction *newTrans, int index)
00047 {
00048     block->transList[index] = *newTrans; /* ye probably we don't need a whole ass function for that*/
00049 }
```

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.](#)

```
00001 int balance(int budget)
00002 {
00003     int unitoCoin_in;
00004     int unitoCoin_out;
00005     int unitoCoin_out_notregistered;
00006     int curr_balance = unitoCoin_in + unitoCoin_out - unitoCoin_out_notregistered;
00007 };
00008
```

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 /* there'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.](#)

```

00001 #ifndef SIMULAZIONE_TRANSAZIONI_COMMON_H
00002 #define SIMULAZIONE_TRANSAZIONI_COMMON_H
00003
00004 #include <stdlib.h>
00005 #include <stdio.h>
00006 #include <stddef.h>
00007 #include <stdint.h>
00008 #include <unistd.h>
00009 #include <errno.h>
00010 #include <time.h>
00011 #include <math.h>
00012 #include <string.h>
00013 #include <sys/ipc.h>
00014 #include <sys/shm.h>
00015 #include <sys/sem.h>
00016 #include <sys/msg.h>
00017 #include <sys/types.h>
00018
00019 #include "../utils/debug.h"
00020 #include "../utils/sem.h"
00021 #include "../utils/lists.h"
00022
00023 #ifndef NULL
00024 #define NULL 0 /* there's a problem with NULL for some reason */
00025 #endif
00026
00027 #ifndef _GNU_SOURCE
00028 #define _GNU_SOURCE
00029 #endif
00030
00031 #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
00032 #define RAND(min, max) ((rand() % (max - min + 1)) + min)
00033
00034 /* -- IPC OBJECTS -- */
00035 #define IPC_ERROR -1
00036 #define SHM_PARAMETERS 1337
00037 #define SHM_USERS_ARRAY 1338
00038 #define SHM_NODES_ARRAY 1339
00039 #define SHM_LEDGER 1340
00040 #define SEM_MASTER 420
00041 #define M_QUEUE_KEY 0x5AD
00042
00043 #define SO_BLOCK_SIZE 100 /* number of transaction per block*/
00044 #define SO_REGISTRY_SIZE 1000 /* max length of consecutive blocks */
00045 #define SELF -1
00046 #define EVERYONE_BROKE '$'
00047
00048 /* -- ARGV LOCATION OF IPC OBJECTS -- */
00049 #define USERS_PID_ARGV (atoi(argv[1]))
00050 #define NODES_PID_ARGV (atoi(argv[2]))
00051 #define PARAMETERS_ARGV (atoi(argv[3]))
00052 #define LEDGER_ARGV (atoi(argv[4]))
00053 #define SEM_ID_ARGV (atoi(argv[5]))
00054
00055 /* -- USER RETURN STATUS -- */
00056 #define WENT_BROKE 1
00057 #define MAX_RETRY 2
00058 #define ERROR -1
00059 #define SUCCESS 0
00060
00061 extern int errno;
00062
00063 #ifndef TEST_ERROR
00064 #define TEST_ERROR
00065     if (errno)
00066     {
00067         fprintf(stderr,
00068             "%s:%d: PID=%5d: Error %d (%s)\n",
00069             __FILE__,
00070             __LINE__,
00071             getpid(),
00072             errno,
00073             strerror(errno));
00074     }
00075 #endif
00076
00077 struct parameters
00078 {
00079     unsigned int SO_USER_NUM;
00080     unsigned int SO_NODES_NUM;
00081     unsigned int SO_BUDGET_INIT;
00082     char SO_REWARD; /* max value 100 */

```

```

00083     unsigned long SO_MIN_TRANS_GEN_NSEC;
00084     unsigned long SO_MAX_TRANS_GEN_NSEC;
00085     unsigned int SO_RETRY;
00086     unsigned int SO_TP_SIZE;
00087     unsigned long SO_MIN_TRANS_PROC_NSEC;
00088     unsigned long SO_MAX_TRANS_PROC_NSEC;
00089     unsigned int SO_SIM_SEC;
00090     unsigned int SO_FRIENDS_NUM;
00091     unsigned int SO_HOPS;
00092 };
00093
00094 typedef struct user_t
00095 {
00096     pid_t pid;
00097     enum
00098     {
00099         alive,
00100         broke,
00101         dead
00102     } status;
00103 } user;
00104
00105 typedef struct node_t
00106 {
00107     pid_t pid;
00108     enum
00109     {
00110         available,
00111         full
00112     } status;
00113 } node;
00114
00115 /* Transaction struct */
00116 typedef struct transaction_t
00117 {
00118     struct timespec timestamp;
00119     pid_t sender;
00120     pid_t receiver;
00121     int amount;
00122     int reward;
00123     enum
00124     {
00125         pending,
00126         processing,
00127         confirmed,
00128         aborted
00129     } status;
00130 } transaction;
00131
00132 struct message
00133 {
00134     long mtype;
00135     transaction userTrans;
00136 };
00137
00138 /* Block struct */
00139 typedef struct block_t
00140 {
00141     transaction transList[SO_BLOCK_SIZE];
00142     unsigned int blockIndex; /* when a block is written on ledger it's Index needs to be updated */
00143     struct block *next;
00144     struct block *prev;
00145 } block;
00146
00147 /* Libro Mastro (ledger) struct */
00148 typedef struct ledger_t
00149 {
00150     block *head;
00151     unsigned int registryCurrSize; /* initialize to SO_REGISTRY_SIZE, update with every new block
added */
00152 } ledger;
00153
00154 ledger *ledger_init();
00155 block *new_block();
00156 void add_block(block);
00157 void add_transaction_to_block(block *, transaction *, int index);
00158 void add_block_to_ledger(block *);
00159 void find_transaction(struct timespec timestamp, pid_t sender, pid_t receiver); /* NULL used to group
results */
00160
00161 /* listparser.c */
00162 void search_timestamp();
00163 void search_sender();
00164 void search_receiver();
00165
00166 #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 *IPCHandle, 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 * IPCHandle,
    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.](#)

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_MASTER_H
00002 #define SIMULAZIONE_TRANSAZIONI_MASTER_H
00003
00004 /*
00005  * probabilmente tutte set macro non ci servono perche' ha più senso passare i
00006  * valori come parametri in compilazione
00007  */
00008
00009 /* libro mastro ----> linked list */
00010
00011 /*
00012  * execve(const char *user, char *const argv[], char *const envp[])
00013  * in user_fork to link "user" executable to the forked process, same thing for nodes
00014  * check lesson on pipes by prof. bini, the second hour
00015  */
00016
00017 /*
00018  * every transaction should be noted and we need to manage inconsistency with
00019  * semaphores maybe
00020  */
00021
```



```

00022 /*
00023  * get every user_pid, ask to libro_mastro
00024  * to return it's current_budget and print
00025  * repeat every second until simulation persists
00026  * remember that CTRL-C should kill the simulation
00027  */
00028
00029 /*
00030  * need to define a kill signal for the simulation, either:
00031  * - SO_SIM_SEC seconds have passed
00032  * - libro_mastro is full
00033  * - CTRL-C from stdin
00034  */
00035
00036 /*
00037  * end of simulation should print:
00038  * - kill signal
00039  * - balance of every user, as before, may need to write a function for that
00040  * - balance of every node (function as before but with different parameter)
00041  * - number of user processes aborted
00042  * - number of blocks in the libro_mastro
00043  * - number of transaction still in the pool, for each node
00044  */
00045
00046 #include <string.h>
00047 #include <signal.h>
00048
00049 void make_arguments(int *IPCarray, char *argv[]);
00050
00051 pid_t spawn_user(char *argv[]);
00052 pid_t spawn_node(char *argv[]);
00053
00054 void shared_memory_objects_init(int *shared_memory_objects_IDs);
00055 void semaphores_init();
00056 void make_ipc_array(int *IPC_objects_IDs);
00057
00058 void master_interrupt_handle(int signum);
00059
00060 #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.](#)

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_PRINT_H
00002 #define SIMULAZIONE_TRANSAZIONI_PRINT_H
00003
00004 #include "common.h"
00005
00006 void print_time_to_die();
00007 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 */
00008 void print_kill_signal(); /* need to
    define, prints reason of termination (simTime elapsed/ledger full/every process terminated) */
00009 void print_user_balance(); /* need to
    define, prints balance of every user */
00010 void print_node_balance(); /* need to
    define, prints balance of every node */
00011 void print_num_aborted(); /* need to
    define, prints num of processes aborted */
00012 void print_num_blocks(); /* need to
    define, prints num of blocks saved in the ledger */
00013 void print_transactions_still_in_pool(); /* need to
    define, prints num of transactions still in the pool of each node */
00014
00015 void final_print(pid_t masterPID, user *usersPID, node *nodesPID, struct parameters *par);
00016 void print_parameters(struct parameters *par);
00017
00018 /* formatting ledger and blocks */
00019 void print_block(FILE *fp, block *b);
00020 void print_transaction(FILE *fp, transaction *t);
00021 void print_ledger(ledger *l);
00022
00023 void formatted_timestamp(FILE *fp);
00024
00025 #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.](#)

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_USERS_H
00002 #define SIMULAZIONE_TRANSAZIONI_USERS_H
00003
00004 #include <signal.h>
00005 #include <stdlib.h>
00006 #include <time.h>
00007 #include <signal.h>
00008
00009 /* sets sleep time with nsec precision for trans_gen */
00010 #define SLEEP_TIME_SET \
00011     randSleepTime.tv_sec = 0; \
00012     randSleepTime.tv_nsec = RAND(par->SO_MIN_TRANS_GEN_NSEC, par->SO_MAX_TRANS_GEN_NSEC);
00013
00014 #define SLEEP \
00015     clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &randSleepTime, &sleepTimeRemaining); \
00016     clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &sleepTimeRemaining, NULL);
00017
00018
00019 void user_transactions_handle(int signum);
00020 void user_interrupt_handle(int signum);
00021
00022 int get_pid_userIndex(int PID_toSearch);
00023 pid_t get_random_userPID();
00024 pid_t get_random_nodePID();
00025
00026 void queue_to_pid(pid_t nodePID);
00027 void transaction_init(pid_t nodePID, int amount, int reward);
00028 int send_transaction();
00029
00030 #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.](#)

```

00001 #include "include/common.h"
00002 #include "include/master.h"
00003 #include "include/print.h"
00004 #include "include/parser.h"
00005
00006 #define SHM_NUM 4
00007 #define SEM_NUM 1
00008 #define IPC_NUM 8
00009
00010 #define USER_NAME "./users"
00011 #define NODE_NAME "./nodes"
00012
00013 /*
00014  =====
00015  || GLOBAL VARIABLES ||
00016  =====
00017  */
00018
00019 /* parameters of simulation */
00020 struct parameters *par;
00021 user *usersPID;
00022 node *nodesPID;
00023 ledger *mainLedger;
00024
00025 int semID;
00026
00027 /*extern int usersPrematurelyDead = 0;*/
00028
00029 /*
00030  =====
00031  || FUNCTIONS ||
00032  =====
00033  */
00034
00035 /* make argv array with IPC IDs for user and nodes */
00036 void make_arguments(int *IPC_array, char **argv)
00037 {
00038     char *uPID_array = malloc(3 * sizeof(IPC_array[0]) + 1);
00039     char *nPID_array = malloc(3 * sizeof(IPC_array[0]) + 1);
00040     char *parameters = malloc(3 * sizeof(IPC_array[0]) + 1);
00041     char *ledger = malloc(3 * sizeof(IPC_array[0]) + 1);
00042     char *semID = malloc(3 * sizeof(IPC_array[0]) + 1);
00043
00044     sprintf(uPID_array, "%d", IPC_array[0]);
00045     sprintf(nPID_array, "%d", IPC_array[1]);
00046     sprintf(parameters, "%d", IPC_array[2]);
00047     sprintf(ledger, "%d", IPC_array[3]);
00048     sprintf(semID, "%d", IPC_array[4]);
00049
00050     argv[0] = USER_NAME; /* need nodes to have a different name but not a priority */
00051     argv[1] = uPID_array;
00052     TRACE(":master: argv[uPID] = %s\n", uPID_array)
00053     argv[2] = nPID_array;
00054     TRACE(":master: argv[nPID] = %s\n", nPID_array)
00055     argv[3] = parameters;
00056     TRACE(":master: argv[par] = %s\n", parameters)
00057     argv[4] = ledger;
00058     TRACE(":master: argv[ledger] = %s\n", ledger)
00059     argv[5] = semID;
00060     TRACE(":master: argv[sem] = %s\n", semID)
00061     argv[8] = NULL; /* Terminating argv with NULL value */
00062 }
00063
00064 /* fork and execve a "./users" */
00065 pid_t spawn_user(char *userArgv[])
00066 {
00067     pid_t myPID = fork();
00068     TRACE(":master: argv values: %s %s %s %s %s %s\n", userArgv[0], userArgv[1], userArgv[2],
userArgv[3], userArgv[4], userArgv[5])
00069     switch (myPID)
00070     {
00071     case -1: /* Error case */
00072         printf("-- Error forking for user\n");
00073         break;
00074
00075     case 0: /* Child case */
00076         TRACE(":master: Spawning user\n");
00077         execve(USER_NAME, userArgv, NULL);
00078         TEST_ERROR
00079         TRACE("!! Message that should never be seen\n");
00080         break;
00081

```

```

00082     default:
00083         return myPID;
00084     }
00085 }
00086
00087 /* fork and execve a "./nodes" */
00088 pid_t spawn_node(char *nodeArgv[])
00089 {
00090     pid_t myPID = fork();
00091     TRACE(("master: argv values: %s %s %s %s %s %s\n", nodeArgv[0], nodeArgv[1], nodeArgv[2],
nodeArgv[3], nodeArgv[4], nodeArgv[5]));
00092     switch (myPID)
00093     {
00094     case -1: /* Error case */
00095         printf("!! Error forking for node\n");
00096         break;
00097
00098     case 0: /* Child case */
00099         TRACE(("master: Spawning node\n"));
00100         execve(NODE_NAME, nodeArgv, NULL);
00101         TEST_ERROR
00102         TRACE(("!! Message that should never be seen\n"));
00103         break;
00104
00105     default:
00106         return myPID;
00107     }
00108 }
00109
00110 /* attach usersPID, nodesPID, par and mainLedger to shared memory, returns an array with respective
IDs */
00111 void shared_memory_objects_init(int *shmArray)
00112 {
00113     /* shared memory segments IDs */
00114     int usersPID_ID;
00115     int nodesPID_ID;
00116     int mainLedger_ID;
00117     int par_ID;
00118
00119     /* parameters init and read from conf file */
00120     par_ID = shmget(SHM_PARAMETERS, sizeof(par), 0600 | IPC_CREAT | IPC_EXCL);
00121     TEST_ERROR
00122     par = (struct parameters *)shmat(par_ID, NULL, 0);
00123     if (parse_parameters(par) == CONF_ERROR)
00124     {
00125         TRACE(("-- Error reading conf file, defaulting to conf#1\n"));
00126     }
00127     else
00128     {
00129         TRACE(("-- Conf file read successful\n"));
00130     }
00131 #ifdef DEBUG
00132     print_parameters(par);
00133 #endif
00134     /* (users_t) and (nodes_t) arrays */
00135     usersPID_ID = shmget(SHM_USERS_ARRAY,
(par->SO_USER_NUM) * sizeof(user),
00136                         0600 | IPC_CREAT | IPC_EXCL);
00137
00138     TEST_ERROR
00139     nodesPID_ID = shmget(SHM_NODES_ARRAY,
(par->SO_NODES_NUM) * sizeof(node),
00140                         0600 | IPC_CREAT | IPC_EXCL);
00141
00142     TEST_ERROR
00143     usersPID = (user *)shmat(usersPID_ID, NULL, 0);
00144     nodesPID = (node *)shmat(nodesPID_ID, NULL, 0);
00145
00146     /* mainLedger */
00147     mainLedger_ID = shmget(SHM_LEDGER,
(par->SO_NODES_NUM) * sizeof(node),
00148                         0600 | IPC_CREAT | IPC_EXCL);
00149
00150     TEST_ERROR
00151     mainLedger = (ledger *)shmat(mainLedger_ID, NULL, 0);
00152
00153     /* mark for deallocation so that they are automatically
* removed once master dies
*/
00156     shmctl(usersPID_ID, IPC_RMID, NULL);
00157     shmctl(nodesPID_ID, IPC_RMID, NULL);
00158     shmctl(par_ID, IPC_RMID, NULL);
00159     shmctl(mainLedger_ID, IPC_RMID, NULL);
00160
00161     shmArray[0] = usersPID_ID;
00162     shmArray[1] = nodesPID_ID;
00163     shmArray[2] = par_ID;
00164     shmArray[3] = mainLedger_ID;
00165 }
00166

```

```

00167 void semaphores_init()
00168 {
00169     semID = semget(SEM_MASTER, 1, 0600 | IPC_CREAT | IPC_EXCL);
00170     TEST_ERROR
00171     TRACE(("master: semID is %d\n", semID));
00172 }
00173
00174 /* makes an array with every IPC object ID */
00175 void make_ipc_array(int *IPC_array)
00176 {
00177     int shmIDs[SHM_NUM]; /* array containing every shared memory ID */
00178     int semIDs[1] = {0};
00179
00180     shared_memory_objects_init(shmIDs);
00181     semIDs[0] = semID;
00182     /* semaphores_init(semIDs); */
00183     memcpy(IPC_array, shmIDs, SHM_NUM * sizeof(int));
00184     memcpy(IPC_array + SHM_NUM, semIDs, SEM_NUM * sizeof(int));
00185 }
00186
00187 /* CTRL-C handler */
00188 void master_interrupt_handle(int signum)
00189 {
00190     write(1, "::Master:: SIGINT ricevuto\n", 28);
00191     killpg(0, SIGINT);
00192
00193     /* just to avoid printing before everyone has finished*/
00194     sleep(1);
00195     final_print(getpid(), usersPID, nodesPID, par);
00196
00197     /*
00198     int status;
00199
00200     while (wait(&status) != -1)
00201     {
00202         status > 8; /* no idea about what it does please help *
00203     }
00204     */
00205
00206     semctl(semID, 1, IPC_RMID);
00207     exit(0);
00208 }
00209
00210 int main(int argc, char *argv[])
00211 {
00212     pid_t myPID = getpid();
00213
00214     int uCounter, nCounter, returnVal;
00215     int simTime;
00216     int ipcObjectsIDs[IPC_NUM];
00217     char **argvSpawns = malloc(8*32);
00218
00219     struct sigaction sa;
00220     struct sembuf sops;
00221
00222     semaphores_init();
00223     make_ipc_array(ipcObjectsIDs);
00224     make_arguments(ipcObjectsIDs, argvSpawns);
00225     mainLedger = ledger_init();
00226
00227     simTime = par->SO_SIM_SEC;
00228
00229     /* -- SIGNAL HANDLER --
00230     * first set all bytes of sigaction to 0
00231     * then initialize sa.handler to a pointer to the function interrupt_handle
00232     * then set the handler to handle SIGINT signals ((struct sigaction *oldact) = NULL)
00233     */
00234     bzero(&sa, sizeof(sa));
00235     sa.sa_handler = master_interrupt_handle;
00236     sigaction(SIGINT, &sa, NULL);
00237
00238     for (nCounter = 0; nCounter < par->SO_NODES_NUM; nCounter++)
00239     {
00240         LOCK
00241         nodesPID[nCounter]
00242             .status = available;
00243         nodesPID[nCounter].pid = spawn_node(argvSpawns);
00244         UNLOCK
00245         if (getpid() != myPID)
00246         {
00247             return;
00248         }
00249     }
00250
00251     /*usersPrematurelyDead = 0;*/
00252     for (uCounter = 0; uCounter < par->SO_USER_NUM; uCounter++)
00253     {

```



```

00254         LOCK
00255         usersPID[uCounter]
00256             .status = alive;
00257         usersPID[uCounter].pid = spawn_user(argvSpawns);
00258         UNLOCK
00259         if (getpid() != myPID)
00260         {
00261             switch (returnVal = wait(NULL))
00262             {
00263                 case MAX_RETRY:
00264                     /* change status in usersPID */
00265                     printf("User %d has died because of too many retries :(\n", getpid());
00266                     break;
00267             }
00268         }
00269         return;
00270     }
00271 }
00272
00273 sleep(simTime);
00274
00275 print_time_to_die();
00276 killpg(0, SIGINT); /* our sigint handler needs to do quite a lot of things to print the wall of
test below */
00277
00278 return 0;
00279 }

```

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     arrayProcessor();
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 arrayProcessor()
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);
```

```
00056 */
00057
00058 int main(int argc, char *argv[])
00059 {
00060     int myPID = getpid();
00061     printf("Node %d has finished\n", myPID);
00062     return 0;
00063
00064     queueID = msgget(myPID, IPC_CREAT | 0600);
00065 }
```

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.](#)

```

00001 #include "include/common.h"
00002 #include "utils/debug.h"
00003 #include "include/parser.h"
00004
00005 /*enum paramID
00006 {
00007     SO_USER_NUM,
00008     SO_NODES_NUM,
00009     SO_NUM_FRIENDS,
00010     SO_SIM_SEC,
00011     SO_HOPS,
00012     SO_BUDGET_INIT,
00013     SO_REWARD,
00014     SO_MIN_TRANS_GEN_NSEC,
00015     SO_MAX_TRANS_GEN_NSEC,
00016     SO_RETRY,
00017     SO_TP_SIZE,
00018     SO_MIN_TRANS_PROC_NSEC,
00019     SO_MAX_TRANS_PROC_NSEC
00020 };
00021
00022 struct parameters
00023 {
00024     char *string;
00025     enum paramID id;
00026 } paramList[] = {
00027     {"SO_USER_NUM", SO_USER_NUM},
00028     {"SO_NODES_NUM", SO_NODES_NUM},
00029     {"SO_NUM_FRIENDS", SO_NUM_FRIENDS},
00030     {"SO_SIM_SEC", SO_SIM_SEC},
00031     {"SO_HOPS", SO_HOPS},
00032     {"SO_BUDGET_INIT", SO_BUDGET_INIT},
00033     {"SO_REWARD", SO_REWARD},
00034     {"SO_MIN_TRANS_GEN_NSEC", SO_MIN_TRANS_GEN_NSEC},
00035     {"SO_MAX_TRANS_GEN_NSEC", SO_MAX_TRANS_GEN_NSEC},
00036     {"SO_RETRY", SO_RETRY},
00037     {"SO_TP_SIZE", SO_TP_SIZE},
00038     {"SO_MIN_TRANS_PROC_NSEC", SO_MIN_TRANS_PROC_NSEC},
00039     {"SO_MAX_TRANS_PROC_NSEC", SO_MAX_TRANS_PROC_NSEC}};*/
00040
00041 void assign_defaults(struct parameters *par)
00042 {
00043     par->SO_USER_NUM = 100;
00044     par->SO_NODES_NUM = 10;
00045     par->SO_BUDGET_INIT = 1000;
00046     par->SO_REWARD = 1;
00047     par->SO_MIN_TRANS_GEN_NSEC = 1000000000;
00048     par->SO_MAX_TRANS_GEN_NSEC = 2000000000;
00049     par->SO_RETRY = 20;
00050     par->SO_TP_SIZE = 1000;
00051     par->SO_MIN_TRANS_PROC_NSEC = 1000000000;
00052     par->SO_MAX_TRANS_PROC_NSEC = 2000000000;
00053     par->SO_SIM_SEC = 10;
00054     par->SO_FRIENDS_NUM = 3;
00055     par->SO_HOPS = 10;
00056 }
00057
00058 int parse_parameters(struct parameters *par)
00059 {
00060     FILE *fp;
00061
00062     /*enum paramID tokensE;*/
00063
00064     /* longer than NUM_PARAMETERS to account for comments and such */
00065     char buffer[128];
00066     int i = 0;
00067
00068     char *tokens[NUM_PARAMETERS];
00069     unsigned long values[NUM_PARAMETERS]; /* downcast is easy, upcast not so much */
00070
00071     /*struct parameters *par = malloc(sizeof(struct parameters));*/
00072     assign_defaults(par);
00073     TRACE(("parser: assigned defaults\n"));
00074
00075     fp = fopen(CONF_FILE, "r");
00076     if (fp == NULL)
00077         return CONF_ERROR; /* default config */
00078
00079     while (fgets(buffer, 127, fp))
00080     {
00081         tokens[i] = malloc(64);
00082         /*values[i] = malloc(sizeof(int));*/

```

```

00083
00084     sscanf(buffer, "%s %lu", tokens[i], &values[i]);
00085
00086     i++;
00087 }
00088
00089 for (i = 0; i < NUM_PARAMETERS; i++)
00090 {
00091     /*switch(tokensE) {
00092         case SO_USER_NUM:
00093         case SO_NODES_NUM:
00094         case SO_NUM_FRIENDS:
00095         case SO_SIM_SEC:
00096         case SO_HOPS:
00097         case SO_BUDGET_INIT:
00098         case SO_REWARD:
00099         case SO_MIN_TRANS_GEN_NSEC:
00100         case SO_MAX_TRANS_GEN_NSEC:
00101         case SO_RETRY:
00102         case SO_TP_SIZE:
00103         case SO_MIN_TRANS_PROC_NSEC:
00104         case SO_MAX_TRANS_PROC_NSEC:
00105         default:
00106             break;
00107     } it can be implemented in a nicer to look at way, but not now */
00108
00109     /*printf("%s\n", tokens[i]);*/
00110
00111     if (!strcmp(tokens[i], "SO_USER_NUM"))
00112     {
00113         par->SO_USER_NUM = values[i];
00114     }
00115     else if (!strcmp(tokens[i], "SO_NODES_NUM"))
00116     {
00117         par->SO_NODES_NUM = values[i];
00118     }
00119     else if (!strcmp(tokens[i], "SO_BUDGET_INIT"))
00120     {
00121         par->SO_BUDGET_INIT = values[i];
00122     }
00123     else if (!strcmp(tokens[i], "SO_REWARD"))
00124     {
00125         /* given that it is a char it's very easy to get it out of bound,
00126          * I prefer straight up normalizing it rather than resetting everything
00127          * because of ERANGE
00128          */
00129         if (values[i] >= 0 && values[i] <= 100)
00130             par->SO_REWARD = values[i];
00131         else
00132             printf(":parser: SO_REWARD incorrect value, resetting default\n");
00133     }
00134     else if (!strcmp(tokens[i], "SO_MIN_TRANS_GEN_NSEC"))
00135     {
00136         par->SO_MIN_TRANS_GEN_NSEC = values[i];
00137     }
00138     else if (!strcmp(tokens[i], "SO_MAX_TRANS_GEN_NSEC"))
00139     {
00140         par->SO_MAX_TRANS_GEN_NSEC = values[i];
00141     }
00142     else if (!strcmp(tokens[i], "SO_RETRY"))
00143     {
00144         par->SO_RETRY = values[i];
00145     }
00146     else if (!strcmp(tokens[i], "SO_TP_SIZE"))
00147     {
00148         par->SO_TP_SIZE = values[i];
00149     }
00150     else if (!strcmp(tokens[i], "SO_MIN_TRANS_PROC_NSEC"))
00151     {
00152         par->SO_MIN_TRANS_PROC_NSEC = values[i];
00153     }
00154     else if (!strcmp(tokens[i], "SO_MAX_TRANS_PROC_NSEC"))
00155     {
00156         par->SO_MAX_TRANS_PROC_NSEC = values[i];
00157     }
00158     else if (!strcmp(tokens[i], "SO_SIM_SEC"))
00159     {
00160         par->SO_SIM_SEC = values[i];
00161     }
00162     else if (!strcmp(tokens[i], "SO_FRIENDS_NUM"))
00163     {
00164         par->SO_FRIENDS_NUM = values[i];
00165     }
00166     else if (!strcmp(tokens[i], "SO_HOPS"))
00167     {
00168         par->SO_HOPS = values[i];
00169     }

```

```

00170     }
00171
00172     /* -- CONF ERRORS CORRECTION -- */
00173     if (errno == ERANGE)
00174     {
00175         TRACE(("parser: one or multiple values out of bound, resetting defaults\n"));
00176         assign_defaults(par);
00177     }
00178     if (par->SO_MIN_TRANS_GEN_NSEC > par->SO_MAX_TRANS_GEN_NSEC)
00179     {
00180         TRACE(("parser: SO_MIN_TRANS_GEN_NSEC greater than SO_MAX_TRANS_GEN_NSEC, will be
normalized\n"));
00181         par->SO_MIN_TRANS_GEN_NSEC = par->SO_MAX_TRANS_GEN_NSEC;
00182     }
00183     if (par->SO_MIN_TRANS_PROC_NSEC > par->SO_MAX_TRANS_PROC_NSEC)
00184     {
00185         TRACE(("parser: SO_MIN_TRANS_PROC_NSEC greater than SO_MAX_TRANS_PROC_NSEC, will be
normalized\n"));
00186         par->SO_MIN_TRANS_PROC_NSEC = par->SO_MAX_TRANS_PROC_NSEC;
00187     }
00188
00189     TRACE(("-----\n----- Configuration input
-----\n"));
00190     for (i = 0; i < NUM_PARAMETERS; i++)
00191     {
00192         TRACE((" %s %lu\n", tokens[i], values[i]));
00193         free(tokens[i]);
00194     }
00195     TRACE(("-----\n"));
00196
00197     fclose(fp);
00198
00199     return 0;
00200 }

```

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,
00024             userPID[userNum].status);
00025     }
00026     printf(" ----- \n");
00027     while (nodesNum--)
00028     {
00029         printf(" | Node          %d          %d          | \n", nodePID[nodesNum].pid,
00030             nodePID[nodesNum].status);
00031     }
00032     printf(" ----- \n");
00033 }
00034 void print_kill_signal();
00035 void print_user_balance();
00036 void print_node_balance();
00037 void print_num_aborted();
00038 void print_num_blocks();
00039 void print_transactions_still_in_pool();
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 }
```

```

00051
00052 void print_parameters(struct parameters *par)
00053 {
00054     printf("-----\n----- Configuration value
00055     -----\n");
00056     printf("SO_USER_NUM->%u\n", par->SO_USER_NUM);
00057     printf("SO_NODES_NUM->%u\n", par->SO_NODES_NUM);
00058     printf("SO_BUDGET_INIT->%u\n", par->SO_BUDGET_INIT);
00059     printf("SO_REWARD->%u\n", par->SO_REWARD);
00060     printf("SO_MIN_TRANS_GEN_NSEC->%lu\n", par->SO_MIN_TRANS_GEN_NSEC);
00061     printf("SO_MAX_TRANS_GEN_NSEC->%lu\n", par->SO_MAX_TRANS_GEN_NSEC);
00062     printf("SO_RETRY->%u\n", par->SO_RETRY);
00063     printf("SO_TP_SIZE->%u\n", par->SO_TP_SIZE);
00064     printf("SO_MIN_TRANS_PROC_NSEC->%lu\n", par->SO_MIN_TRANS_PROC_NSEC);
00065     printf("SO_MAX_TRANS_PROC_NSEC->%lu\n", par->SO_MAX_TRANS_PROC_NSEC);
00066     printf("SO_SIM_SEC->%u\n", par->SO_SIM_SEC);
00067     printf("SO_FRIENDS_NUM->%u\n", par->SO_FRIENDS_NUM);
00068     printf("SO_HOPS->%u\n", par->SO_HOPS);
00069     printf("-----\n");
00070 }
00071 /*void print_kill_signal(mainPID, userPid /* other process */)
00072 {
00073     printf("-----PROCESS PID NUM %d KILL-----", mainPID);
00074 }
00075 void print_user_balance(int balance)
00076 {
00077     printf("-----CURRENT BALANCE IS:%d-----", balance);
00078 }
00079
00080 void print_node_balance(int balance)
00081 {
00082     printf("-----CURRENT NODE BALANCE IS:%d", balance);
00083 } /*
00084 /*void print_num_aborted()
00085 {
00086     printf("\n-- Num of aborted users: %d\n", usersPrematurelyDead);
00087 }
00088 /*
00089 void print_num_blocks()
00090 {
00091     printf("---TOTAL BLOCK:%d");
00092 }
00093 void print_transactions_still_in_pool()
00094 {
00095     printf("-----TOTAL TRANSACTION STILL IN POLL:%d-----");
00096 }*/
00097
00098 void print_transaction(FILE *fp, transaction *t)
00099 {
00100     char tmp[10];
00101     switch (t->status)
00102     {
00103     case pending:
00104         strcpy(tmp, "pending");
00105         break;
00106     case aborted:
00107         strcpy(tmp, "aborted");
00108         break;
00109     case confirmed:
00110         strcpy(tmp, "confirmed");
00111         break;
00112     case processing:
00113         strcpy(tmp, "confirmed");
00114         break;
00115     }
00116
00117     fprintf(fp, " ----- \n");
00118     formatted_timestamp(fp);
00119     fprintf(fp, " %s", tmp);
00120     fprintf(fp, "| %d --> %d\n", t->sender, t->receiver);
00121     fprintf(fp, "| Amount: %d\n", t->amount);
00122     fprintf(fp, "| Reward: %d\n", t->reward);
00123     fprintf(fp, "| Reward: %d\n", t->reward);
00124     fprintf(fp, " ----- \n");
00125 }
00126
00127 void print_block(FILE *fp, block *b)
00128 {
00129     int i;
00130     block *curr;
00131
00132     for (curr = b; curr != NULL; curr = (block*)curr->next)
00133     {
00134         fprintf(fp, "= %3d =====\n", b->blockIndex);
00135         for (i = 0; i < SO_BLOCK_SIZE; i++)
00136             {

```

```

00137         print_transaction(fp, &(b->transList[i]));
00138     }
00139     fprintf(fp, "=====\n");
00140 }
00141 }
00142
00143 void print_ledger(ledger *l)
00144 {
00145     FILE *fp = fopen("ledger.txt", "w");
00146     if (fp == NULL)
00147     {
00148         printf(":print: couldn't open file pointer ledger.txt\n");
00149     }
00150
00151     fprintf(fp, "Registry Real Size is %d blocks\n", l->registryCurrSize);
00152     print_block(fp, l->head);
00153     fclose(fp);
00154 }
00155
00156 /* print without /n */
00157 void formatted_timestamp(FILE *fp)
00158 {
00159     printf("Hey");
00160     /*clock_t tic = clock();
00161     clock_t start = clock();
00162     clock_t stop = clock();
00163
00164     time_t rawtime;
00165     time_t now;
00166     struct tm *info;
00167     struct tm *today;
00168     double elapsed;
00169     char buf[128];
00170
00171     time(&now);
00172     today = localtime(&now);
00173     strftime(buf, 128, "%Y/%m/%d", today);
00174     printf("%s\n", buf);
00175
00176     elapsed = (double)(stop - start) * 1000.0 / CLOCKS_PER_SEC; /* time ./a.out*/
00177 }

```

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 * saUSRI,
    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;
```

```

00041 int outGoingTransactions; /* accumulate amount of transactions sent but yet to be received */
00042 transaction currTrans;
00043
00044 /*
00045 =====
00046 ||    FUNCTIONS    ||
00047 =====
00048 */
00049
00050 /* returns index of where user with PID_toSearch is located in usersPID[] */
00051 int get_pid_userIndex(int PID_toSearch)
00052 {
00053     int i;
00054
00055     for (i = 0; i < par->SO_USER_NUM; i++)
00056     {
00057         if (usersPID[i].pid == myPID)
00058             return i;
00059     }
00060
00061     return -1;
00062 }
00063
00064 /* returns a random PID of a non-dead user from usersPID[] */
00065 pid_t get_random_userPID()
00066 {
00067     int index;
00068     pid_t val = 0;
00069
00070     do
00071     {
00072         index = RAND(0, par->SO_USER_NUM - 1);
00073         TRACE(("user: %d index is %d\n", myPID, index))
00074         TRACE(("users: %d usersPID[%d]\n", myPID, index));
00075         if (usersPID[index].status != dead)
00076             val = usersPID[index].pid;
00077     } while (!val);
00078
00079     return val;
00080 }
00081
00082 /* returns a random PID of an available node from nodesPID[] */
00083 pid_t get_random_nodePID()
00084 {
00085     int index;
00086     pid_t val = 0;
00087
00088     do
00089     {
00090         index = RAND(0, par->SO_NODES_NUM - 1);
00091         TRACE(("user: %d index is %d\n", myPID, index))
00092         TRACE(("users: %d nodesPID[%d]\n", myPID, index));
00093         if (nodesPID[index].status == available)
00094             val = nodesPID[index].pid;
00095     } while (!val);
00096
00097     return val;
00098 }
00099
00100 /* safely updates status of user to statusToSet: 0 alive, 1 broke, 2 dead */
00101 void update_status(int statusToSet)
00102 {
00103     int i = get_pid_userIndex(myPID);
00104     if (i == -1)
00105     {
00106         TRACE(("users: %d failed to find myself in usersPID[]", myPID));
00107     }
00108
00109     sem_reserve(semID, 1);
00110     usersPID[i].status = statusToSet;
00111     if (statusToSet == 2)
00112     {
00113         /*usersPrematurelyDead++;*/
00114         TRACE(("users: dead increased\n"));
00115     }
00116     sem_release(semID, 1);
00117 }
00118
00119 /* attaches ipc objects based on IDs passed via arguments */
00120 void attach_ipc_objects(char **argv)
00121 {
00122     par = shmat(PARAMETERS_ARGV, NULL, 0);
00123     TRACE(("users %d par->SO_RETRY %d\n", myPID, par->SO_RETRY))
00124     TEST_ERROR
00125     usersPID = shmat(USERS_PID_ARGV, NULL, 0);
00126     TRACE(("users: %d usersPID[0] = %d, usersPID[3] = %d\n", myPID, usersPID[0], usersPID[3]))
00127     TEST_ERROR

```

```

00128     nodesPID = shmat(NODES_PID_ARGV, NULL, 0);
00129     TRACE(("users: %d nodesPID[0] = %d, nodesPID[3] = %d\n", myPID, nodesPID[0], nodesPID[3]));
00130     TEST_ERROR
00131     mainLedger = shmat(LEDGER_ARGV, NULL, 0);
00132     TEST_ERROR
00133     semID = SEM_ID_ARGV;
00134     TRACE(("users: %d semID is %d\n", myPID, semID));
00135 }
00136
00137 /* use nodePID as key for msgget and check for errors */
00138 void queue_to_pid(pid_t nodePID)
00139 {
00140     queueID = msgget(nodePID, IPC_CREAT | 0600);
00141     TEST_ERROR
00142     TRACE(("users: %d -> %d queueID %d\n", myPID, nodePID, queueID))
00143 }
00144
00145 /* initializes transaction values and sets it to pending */
00146 void transaction_init(pid_t userPID, int amount, int reward)
00147 {
00148     struct timespec exactTime;
00149
00150     currTrans.sender = myPID;
00151     currTrans.receiver = userPID;
00152     currTrans.amount = amount;
00153     currTrans.reward = reward;
00154     currTrans.status = pending;
00155     clock_gettime(CLOCK_REALTIME, &exactTime);
00156     currTrans.timestamp = exactTime;
00157 }
00158
00159 /* initializes signal handlers for SIGINT and SIGUSR1 */
00160 void signal_handlers_init(struct sigaction *saUSR1, struct sigaction *saINT)
00161 {
00162     /* -- SIGNAL HANDLERS --
00163     * first set all bytes of sigaction to 0
00164     * then initialize sa.handler to a pointer to
00165     * the function user_transaction/interrupt_handle
00166     * then set the handler to handle SIUSR1/SIGINT signals
00167     * ((struct sigaction *oldact) = NULL)
00168     */
00169     saUSR1->sa_handler = user_transactions_handle;
00170     saINT->sa_handler = user_interrupt_handle;
00171     sigaction(SIGUSR1, saUSR1, NULL);
00172     sigaction(SIGINT, saINT, NULL);
00173 }
00174
00175 /* send transaction currTrans to user userPID via node nodePID */
00176 int send_transaction()
00177 {
00178     msgsnd(queueID, &currTrans, sizeof(transaction), 0);
00179     TEST_ERROR
00180     currBalance -= (currTrans.amount + currTrans.reward);
00181     outgoingTransactions += (currTrans.amount + currTrans.reward);
00182     switch (errno)
00183     {
00184     case EACCES:
00185         printf("users %d no write permission on queue\n", myPID);
00186         break;
00187     case EAGAIN:
00188         printf("users: %d queue full\n", myPID); /* keep if we decide to use IPC_NOWAIT */
00189         break;
00190     case EFAULT:
00191         printf("users: %d address pointed by msgp inaccessible\n", myPID);
00192         break;
00193     case EIDRM:
00194         printf("users: %d message queue removed\n", myPID);
00195         break;
00196     case EINTR:
00197         TRACE(("users: %d signal caught when waiting for queue to free\n", myPID));
00198         break;
00199     case EINVAL:
00200         printf("users: %d invalid msqid value, or nonpositive mtype value, or invalid msgsz
value\n", myPID);
00201         break;
00202     case ENOMEM:
00203         printf("users: %d system out of memory\n", myPID); /* should basically never happen I hope */
00204         break;
00205     default:
00206         TRACE(("Transaction sent\n"))
00207         return SUCCESS;
00208     }
00209     currTrans.status = aborted;
00210     currBalance += (currTrans.amount + currTrans.reward);
00211     outgoingTransactions -= (currTrans.amount + currTrans.reward);
00212     /* we can then track this type of aborted transactions but rn there's no need to */
00213     return ERROR;

```

```

00214 }
00215
00216 /* SIGUSR1 handler, sends a transaction */
00217 void user_transactions_handle(int signum)
00218 {
00219     write(1, "::User:: SIGUSR1 received\n", 27);
00220     if (currBalance > 2)
00221         send_transaction(); /* we're calling a printf which is not thread safe, need to fix somehow*/
00222     else
00223         write(1, "::User:: sorry balance too low\n", 32);
00224 }
00225
00226 /* CTRL-C handler */
00227 void user_interrupt_handle(int signum)
00228 {
00229     write(1, "::User:: SIGINT received\n", 26);
00230     exit(0);
00231 }
00232
00233 int main(int argc, char *argv[])
00234 {
00235     int amount, reward, retry;
00236     pid_t userPID, nodePID;
00237
00238     struct timespec randSleepTime;
00239     struct timespec sleepTimeRemaining;
00240
00241     struct sembuf sops;
00242     struct message transMsg;
00243
00244     struct sigaction saUSR1;
00245     struct sigaction saINT;
00246     bzero(&saUSR1, sizeof(saUSR1));
00247     bzero(&saINT, sizeof(saINT));
00248
00249     myPID = getpid(); /* set myPID value */
00250     TRACE(":users: %d USERS_PID_ARGV %d\n", myPID, USERS_PID_ARGV)
00251     TRACE(":users: %d NODES_PID_ARGV %d\n", myPID, NODES_PID_ARGV)
00252     TRACE(":users: %d PARAMETERS_ARGV %d\n", myPID, PARAMETERS_ARGV)
00253     TRACE(":users: %d LEDGER_ARGV %d\n", myPID, LEDGER_ARGV)
00254     TRACE(":users: %d SEM_ID_PID_ARGV %d\n", myPID, SEM_ID_ARGV)
00255
00256     if (argc == 0)
00257     {
00258         printf(":users: %d, no arguments passed, can't continue like this any more :C\n", myPID);
00259         return ERROR;
00260     }
00261
00262     srand(time(NULL)); /* initialize rand function */
00263
00264     attach_ipc_objects(argv);
00265     signal_handlers_init(&saUSR1, &saINT);
00266     transMsg.mtype = atol("transaction");
00267
00268     retry = par->SO_RETRY;
00269     while (1)
00270     {
00271         SLEEP_TIME_SET
00272         /*
00273          * save the time unslept when interrupted by SIGUSR1
00274          * so that we can't force transactions at a much greater speed
00275          * better to save it into a separate struct because clock_nanosleep
00276          * will not update it if the sleep is not interrupted
00277          */
00278         bzero(&sleepTimeRemaining, sizeof(sleepTimeRemaining));
00279
00280         currBalance = 100 /*balance(myPID)*/;
00281         if (currBalance >= 2)
00282         {
00283             userPID = get_random_userPID();
00284             nodePID = get_random_nodePID();
00285
00286             amount = RAND(2, currBalance);
00287             reward = REWARD(amount, par->SO_REWARD);
00288             amount -= reward;
00289
00290             queue_to_pid(nodePID);
00291             transaction_init(userPID, amount, reward);
00292             if (send_transaction() == 0)
00293                 retry = par->SO_RETRY;
00294             else
00295                 retry--;
00296
00297             if (retry == 0)
00298             {
00299                 update_status(2);
00300                 return MAX_RETRY;

```

```

00301         }
00302         SLEEP
00303     }
00304     else
00305     {
00306         printf(":users: %d went broke :/\n", myPID);
00307         update_status(1);
00308
00309         /*wait_for_incoming_transaction(); ////////// */
00310     }
00311 }
00312 }

```

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=f3d84f0e4a08461
    */
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/utls/lists.c File Reference

4.32 lists.c

[Go to the documentation of this file.](#)

4.33 src/utls/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/utls/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 }
```

```

00045     return semop(sem_id, &sops, 1);
00046 }
00047
00048 /* Print all semaphore values to a string */
00049 int sem_getall(char *my_string, int sem_id)
00050 {
00051     union semun arg; /* man semctl per vedere def della union */
00052     unsigned short *sem_vals, i;
00053     unsigned long num_sem;
00054     char cur_str[10];
00055     struct semid_ds my_ds;
00056
00057     /* Get the number of semaphores */
00058     arg.buf = &my_ds;
00059     semctl(sem_id, 0, IPC_STAT, arg);
00060     TEST_ERROR;
00061     num_sem = arg.buf->sem_nsems;
00062
00063     /* Get the values of all semaphores */
00064     sem_vals = malloc(sizeof(*sem_vals) * num_sem);
00065     arg.array = sem_vals;
00066     semctl(sem_id, 0, GETALL, arg);
00067
00068     /* Initialize the string. MUST be allocated by the caller */
00069     my_string[0] = 0;
00070     for (i = 0; i < num_sem; i++)
00071     {
00072         sprintf(cur_str, "%d ", sem_vals[i]);
00073         strcat(my_string, cur_str);
00074     }
00075 }

```

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.](#)

```
00001 #ifndef SIMULAZIONE_TRANSAZIONI_SEM_H
00002 #define SIMULAZIONE_TRANSAZIONI_SEM_H
00003
00004 #include <sys/types.h>
00005 #include <sys/ipc.h>
00006 #include <sys/sem.h>
00007 #include <string.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <errno.h>
00011 #include <unistd.h>
00012
00013 /* from prof. Schifanella examples */
00014 #define LOCK \
00015     sops.sem_num = 1; \
00016     sops.sem_op = -1; \
00017     sops.sem_flg = 0; \
00018     semop(semID, &sops, 1);
00019 #define UNLOCK \
00020     sops.sem_num = 1; \
00021     sops.sem_op = 1; \
00022     sops.sem_flg = 0; \
00023     semop(semID, &sops, 1);
00024
00025 /* from prof. Bini examples */
00026
00027 /*
00028  * The following union must be defined as required by the semctl man
00029  * page
00030  */
00031 union semun {
00032     int val; /* Value for SETVAL */
00033     struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
00034     unsigned short *array; /* Array for GETALL, SETALL */
00035     struct seminfo *__buf; /* Buffer for IPC_INFO
00036                             (Linux-specific) */
00037 };
00038
00039
00040 /*
00041  * Set a semaphore to a user defined value
00042  * INPUT:
00043  * - sem_id: the ID of the semaphore IPC object
00044  * - sem_num: the position of the semaphore in the array
00045  * - sem_val: the initialization value of the semaphore
00046  * RESULT:
00047  * - the selected semaphore is initialized to the given value
```

```
00048  * - the returned value is the same as the invoked semctl
00049  */
00050  int sem_set_val(int sem_id, int sem_num, int sem_val);
00051
00052  /*
00053  * Try to access the resource
00054  * INPUT:
00055  * - sem_id: the ID of the semaphore IPC object
00056  * - sem_num: the position of the semaphore in the array
00057  * RESULT
00058  * - if the resource is available (semaphore value > 0), the semaphore
00059  *   is decremented by one
00060  * - if the resource is not available (semaphore value == 0), the
00061  *   process is blocked until the resource becomes available again
00062  * - the returned value is the same as the invoked semop
00063  */
00064  int sem_reserve(int sem_id, int sem_num);
00065
00066  /*
00067  * Release the resource
00068  * INPUT:
00069  * - sem_id: the ID of the semaphore IPC object
00070  * - sem_num: the position of the semaphore in the array
00071  * RESULT:
00072  * - the semaphore value is incremented by one. This may unblock some
00073  *   process
00074  * - the returned value is the same as the invoked semop
00075  */
00076  int sem_release(int sem_id, int sem_num);
00077
00078  /*
00079  * Print all semaphore values to a string. my_string MUST be
00080  * previously allocated
00081  */
00082  int sem_getall(char * my_string, int sem_id);
00083
00084
00085  #endif /* SIMULAZIONE_TRANSAZIONI_SEM_H */
```


Index

- [_GNU_SOURCE](#)
 - [common.h, 20](#)
 - [__buf](#)
 - [semun, 12](#)
- [aborted](#)
 - [transaction_t, 14](#)
- [add_block](#)
 - [common.h, 25](#)
- [add_block_to_ledger](#)
 - [common.h, 25](#)
- [add_transaction_to_block](#)
 - [common.h, 25](#)
- [alive](#)
 - [user_t, 16](#)
- [amount](#)
 - [transaction_t, 14](#)
- [array](#)
 - [semun, 12](#)
- [ARRAY_SIZE](#)
 - [common.h, 20](#)
- [assign_defaults](#)
 - [parser.c, 50](#)
 - [parser.h, 33](#)
- [attach_ipc_objects](#)
 - [users.c, 59](#)
- [available](#)
 - [node_t, 8](#)
- [balance](#)
 - [balance.h, 18](#)
- [balance.h](#)
 - [balance, 18](#)
- [block](#)
 - [common.h, 24](#)
- [block_t, 5](#)
 - [blockIndex, 5](#)
 - [next, 5](#)
 - [prev, 6](#)
 - [transList, 6](#)
- [blockIndex](#)
 - [block_t, 5](#)
- [broke](#)
 - [user_t, 16](#)
- [buf](#)
 - [semun, 13](#)
- [common.h](#)
 - [_GNU_SOURCE, 20](#)
 - [add_block, 25](#)
 - [add_block_to_ledger, 25](#)
 - [add_transaction_to_block, 25](#)
 - [ARRAY_SIZE, 20](#)
 - [block, 24](#)
 - [errno, 26](#)
 - [ERROR, 20](#)
 - [EVERYONE_BROKE, 20](#)
 - [find_transaction, 25](#)
 - [IPC_ERROR, 20](#)
 - [ledger, 24](#)
 - [LEDGER_ARGV, 20](#)
 - [ledger_init, 25](#)
 - [M_QUEUE_KEY, 21](#)
 - [MAX_RETRY, 21](#)
 - [new_block, 25](#)
 - [node, 24](#)
 - [NODES_PID_ARGV, 21](#)
 - [NULL, 21](#)
 - [PARAMETERS_ARGV, 21](#)
 - [RAND, 21](#)
 - [search_receiver, 26](#)
 - [search_sender, 26](#)
 - [search_timestamp, 26](#)
 - [SELF, 22](#)
 - [SEM_ID_ARGV, 22](#)
 - [SEM_MASTER, 22](#)
 - [SHM_LEDGER, 22](#)
 - [SHM_NODES_ARRAY, 22](#)
 - [SHM_PARAMETERS, 22](#)
 - [SHM_USERS_ARRAY, 23](#)
 - [SO_BLOCK_SIZE, 23](#)
 - [SO_REGISTRY_SIZE, 23](#)
 - [SUCCESS, 23](#)
 - [TEST_ERROR, 23](#)
 - [transaction, 24](#)
 - [user, 24](#)
 - [USERS_PID_ARGV, 23](#)
 - [WENT_BROKE, 24](#)
- [CONF_ERROR](#)
 - [parser.h, 32](#)
- [CONF_FILE](#)
 - [parser.h, 32](#)
- [confirmed](#)
 - [transaction_t, 14](#)
- [currBalance](#)
 - [users.c, 61](#)
- [currTrans](#)
 - [users.c, 61](#)
- [dbg_printf](#)

- debug.c, [67](#)
 - debug.h, [68](#)
- dead
 - user_t, [16](#)
- debug.c
 - dbg_printf, [67](#)
- debug.h
 - dbg_printf, [68](#)
 - TRACE, [68](#)
- errno
 - common.h, [26](#)
 - sem.c, [71](#)
- ERROR
 - common.h, [20](#)
- EVERYONE_BROKE
 - common.h, [20](#)
- final_print
 - print.c, [53](#)
 - print.h, [34](#)
- find_transaction
 - common.h, [25](#)
- formatted_timestamp
 - print.c, [54](#)
 - print.h, [34](#)
- full
 - node_t, [8](#)
- get_pid_userIndex
 - users.c, [59](#)
 - users.h, [38](#)
- get_random_nodePID
 - users.c, [59](#)
 - users.h, [38](#)
- get_random_userPID
 - users.c, [60](#)
 - users.h, [38](#)
- head
 - ledger_t, [6](#)
- IPC_ERROR
 - common.h, [20](#)
- IPC_NUM
 - master.c, [40](#)
- ledger
 - common.h, [24](#)
- LEDGER_ARGV
 - common.h, [20](#)
- ledger_init
 - common.h, [25](#)
- ledger_t, [6](#)
 - head, [6](#)
 - registryCurrSize, [7](#)
- LOCK
 - sem.h, [73](#)
- M_QUEUE_KEY
 - common.h, [21](#)
- main
 - master.c, [41](#)
 - nodes.c, [47](#)
 - users.c, [60](#)
- mainLedger
 - master.c, [43](#)
 - nodes.c, [48](#)
 - users.c, [62](#)
- make_arguments
 - master.c, [41](#)
 - master.h, [29](#)
- make_ipc_array
 - master.c, [42](#)
 - master.h, [29](#)
- master.c
 - IPC_NUM, [40](#)
 - main, [41](#)
 - mainLedger, [43](#)
 - make_arguments, [41](#)
 - make_ipc_array, [42](#)
 - master_interrupt_handle, [42](#)
 - NODE_NAME, [40](#)
 - nodesPID, [43](#)
 - par, [43](#)
 - SEM_NUM, [41](#)
 - semaphores_init, [42](#)
 - semID, [43](#)
 - shared_memory_objects_init, [42](#)
 - SHM_NUM, [41](#)
 - spawn_node, [42](#)
 - spawn_user, [42](#)
 - USER_NAME, [41](#)
 - usersPID, [43](#)
- master.h
 - make_arguments, [29](#)
 - make_ipc_array, [29](#)
 - master_interrupt_handle, [29](#)
 - semaphores_init, [29](#)
 - shared_memory_objects_init, [30](#)
 - spawn_node, [30](#)
 - spawn_user, [30](#)
- master_interrupt_handle
 - master.c, [42](#)
 - master.h, [29](#)
- MAX_RETRY
 - common.h, [21](#)
- message, [7](#)
 - mtype, [7](#)
 - userTrans, [7](#)
- mtype
 - message, [7](#)
- myPID
 - nodes.c, [48](#)
 - users.c, [62](#)
- new_block
 - common.h, [25](#)
- next

- block_t, 5
- node
 - common.h, 24
- NODE_NAME
 - master.c, 40
- node_t, 8
 - available, 8
 - full, 8
 - pid, 9
 - status, 9
- nodes.c
 - main, 47
 - mainLedger, 48
 - myPID, 48
 - nodesPID, 48
 - par, 48
 - queueID, 48
 - semID, 48
 - usersPID, 49
- nodes.h
 - randSleepTime, 31
 - sleepTimeRemaining, 31
- NODES_PID_ARGV
 - common.h, 21
- nodesPID
 - master.c, 43
 - nodes.c, 48
 - users.c, 62
- NULL
 - common.h, 21
- NUM_PARAMETERS
 - parser.h, 32
- outGoingTransactions
 - users.c, 62
- par
 - master.c, 43
 - nodes.c, 48
 - users.c, 62
- parameters, 9
 - SO_BUDGET_INIT, 10
 - SO_FRIENDS_NUM, 10
 - SO_HOPS, 10
 - SO_MAX_TRANS_GEN_NSEC, 10
 - SO_MAX_TRANS_PROC_NSEC, 10
 - SO_MIN_TRANS_GEN_NSEC, 10
 - SO_MIN_TRANS_PROC_NSEC, 11
 - SO_NODES_NUM, 11
 - SO_RETRY, 11
 - SO_REWARD, 11
 - SO_SIM_SEC, 11
 - SO_TP_SIZE, 11
 - SO_USER_NUM, 12
- PARAMETERS_ARGV
 - common.h, 21
- parse_parameters
 - parser.c, 50
 - parser.h, 33
- parser.c
 - assign_defaults, 50
 - parse_parameters, 50
- parser.h
 - assign_defaults, 33
 - CONF_ERROR, 32
 - CONF_FILE, 32
 - NUM_PARAMETERS, 32
 - parse_parameters, 33
- pending
 - transaction_t, 14
- pid
 - node_t, 9
 - user_t, 16
- prev
 - block_t, 6
- print.c
 - final_print, 53
 - formatted_timestamp, 54
 - print_block, 54
 - print_kill_signal, 54
 - print_ledger, 54
 - print_node_balance, 54
 - print_num_aborted, 54
 - print_num_blocks, 55
 - print_parameters, 55
 - print_time_to_die, 55
 - print_transaction, 55
 - print_transactions_still_in_pool, 55
 - print_user_balance, 55
 - print_user_nodes_table, 56
- print.h
 - final_print, 34
 - formatted_timestamp, 34
 - print_block, 34
 - print_kill_signal, 34
 - print_ledger, 35
 - print_node_balance, 35
 - print_num_aborted, 35
 - print_num_blocks, 35
 - print_parameters, 35
 - print_time_to_die, 35
 - print_transaction, 35
 - print_transactions_still_in_pool, 36
 - print_user_balance, 36
 - print_user_nodes_table, 36
- print_block
 - print.c, 54
 - print.h, 34
- print_kill_signal
 - print.c, 54
 - print.h, 34
- print_ledger
 - print.c, 54
 - print.h, 35
- print_node_balance
 - print.c, 54
 - print.h, 35

- print_num_aborted
 - print.c, 54
 - print.h, 35
- print_num_blocks
 - print.c, 55
 - print.h, 35
- print_parameters
 - print.c, 55
 - print.h, 35
- print_time_to_die
 - print.c, 55
 - print.h, 35
- print_transaction
 - print.c, 55
 - print.h, 35
- print_transactions_still_in_pool
 - print.c, 55
 - print.h, 36
- print_user_balance
 - print.c, 55
 - print.h, 36
- print_user_nodes_table
 - print.c, 56
 - print.h, 36
- processing
 - transaction_t, 14
- queue_to_pid
 - users.c, 60
 - users.h, 38
- queueID
 - nodes.c, 48
 - users.c, 62
- RAND
 - common.h, 21
- randSleepTime
 - nodes.h, 31
- receiver
 - transaction_t, 14
- registryCurrSize
 - ledger_t, 7
- REWARD
 - users.c, 59
- reward
 - transaction_t, 14
- search_receiver
 - common.h, 26
- search_sender
 - common.h, 26
- search_timestamp
 - common.h, 26
- SELF
 - common.h, 22
- sem.c
 - errno, 71
 - sem_getall, 70
 - sem_release, 70
 - sem_reserve, 70
 - sem_set_val, 70
 - TEST_ERROR, 69
- sem.h
 - LOCK, 73
 - sem_getall, 73
 - sem_release, 73
 - sem_reserve, 73
 - sem_set_val, 74
 - UNLOCK, 73
- sem_getall
 - sem.c, 70
 - sem.h, 73
- SEM_ID_ARGV
 - common.h, 22
- SEM_MASTER
 - common.h, 22
- SEM_NUM
 - master.c, 41
- sem_release
 - sem.c, 70
 - sem.h, 73
- sem_reserve
 - sem.c, 70
 - sem.h, 73
- sem_set_val
 - sem.c, 70
 - sem.h, 74
- semaphores_init
 - master.c, 42
 - master.h, 29
- semID
 - master.c, 43
 - nodes.c, 48
 - users.c, 63
- semun, 12
 - __buf, 12
 - array, 12
 - buf, 13
 - val, 13
- send_transaction
 - users.c, 60
 - users.h, 38
- sender
 - transaction_t, 14
- shared_memory_objects_init
 - master.c, 42
 - master.h, 30
- SHM_LEDGER
 - common.h, 22
- SHM_NODES_ARRAY
 - common.h, 22
- SHM_NUM
 - master.c, 41
- SHM_PARAMETERS
 - common.h, 22
- SHM_USERS_ARRAY
 - common.h, 23

- signal_handlers_init
 - users.c, 60
- SLEEP
 - users.h, 37
- SLEEP_TIME_SET
 - users.h, 37
- sleepTimeRemaining
 - nodes.h, 31
- SO_BLOCK_SIZE
 - common.h, 23
- SO_BUDGET_INIT
 - parameters, 10
- SO_FRIENDS_NUM
 - parameters, 10
- SO_HOPS
 - parameters, 10
- SO_MAX_TRANS_GEN_NSEC
 - parameters, 10
- SO_MAX_TRANS_PROC_NSEC
 - parameters, 10
- SO_MIN_TRANS_GEN_NSEC
 - parameters, 10
- SO_MIN_TRANS_PROC_NSEC
 - parameters, 11
- SO_NODES_NUM
 - parameters, 11
- SO_REGISTRY_SIZE
 - common.h, 23
- SO_RETRY
 - parameters, 11
- SO_REWARD
 - parameters, 11
- SO_SIM_SEC
 - parameters, 11
- SO_TP_SIZE
 - parameters, 11
- SO_USER_NUM
 - parameters, 12
- spawn_node
 - master.c, 42
 - master.h, 30
- spawn_user
 - master.c, 42
 - master.h, 30
- src/common.c, 17
- src/include/balance.h, 18
- src/include/common.h, 18, 27
- src/include/master.h, 29, 30
- src/include/nodes.h, 31, 32
- src/include/parser.h, 32, 33
- src/include/print.h, 33, 36
- src/include/users.h, 37, 39
- src/master.c, 40, 44
- src/nodes.c, 47, 49
- src/parser.c, 50, 51
- src/print.c, 53, 56
- src/users.c, 58, 63
- src/utls/debug.c, 67
- src/utls/debug.h, 67, 68
- src/utls/lists.c, 69
- src/utls/lists.h, 69
- src/utls/sem.c, 69, 71
- src/utls/sem.h, 72, 74
- status
 - node_t, 9
 - transaction_t, 14
 - user_t, 16
- SUCCESS
 - common.h, 23
- TEST_ERROR
 - common.h, 23
 - sem.c, 69
- timestamp
 - transaction_t, 15
- TRACE
 - debug.h, 68
- transaction
 - common.h, 24
- transaction_init
 - users.c, 60
 - users.h, 38
- transaction_t, 13
 - aborted, 14
 - amount, 14
 - confirmed, 14
 - pending, 14
 - processing, 14
 - receiver, 14
 - reward, 14
 - sender, 14
 - status, 14
 - timestamp, 15
- transList
 - block_t, 6
- UNLOCK
 - sem.h, 73
- update_status
 - users.c, 61
- user
 - common.h, 24
- user_interrupt_handle
 - users.c, 61
 - users.h, 39
- USER_NAME
 - master.c, 41
- user_t, 15
 - alive, 16
 - broke, 16
 - dead, 16
 - pid, 16
 - status, 16
- user_transactions_handle
 - users.c, 61
 - users.h, 39
- users.c

- attach_ipc_objects, [59](#)
- currBalance, [61](#)
- currTrans, [61](#)
- get_pid_userIndex, [59](#)
- get_random_nodePID, [59](#)
- get_random_userPID, [60](#)
- main, [60](#)
- mainLedger, [62](#)
- myPID, [62](#)
- nodesPID, [62](#)
- outGoingTransactions, [62](#)
- par, [62](#)
- queue_to_pid, [60](#)
- queueID, [62](#)
- REWARD, [59](#)
- semID, [63](#)
- send_transaction, [60](#)
- signal_handlers_init, [60](#)
- transaction_init, [60](#)
- update_status, [61](#)
- user_interrupt_handle, [61](#)
- user_transactions_handle, [61](#)
- usersPID, [63](#)
- users.h
 - get_pid_userIndex, [38](#)
 - get_random_nodePID, [38](#)
 - get_random_userPID, [38](#)
 - queue_to_pid, [38](#)
 - send_transaction, [38](#)
 - SLEEP, [37](#)
 - SLEEP_TIME_SET, [37](#)
 - transaction_init, [38](#)
 - user_interrupt_handle, [39](#)
 - user_transactions_handle, [39](#)
- USERS_PID_ARGV
 - common.h, [23](#)
- usersPID
 - master.c, [43](#)
 - nodes.c, [49](#)
 - users.c, [63](#)
- userTrans
 - message, [7](#)
- val
 - semun, [13](#)
- WENT_BROKE
 - common.h, [24](#)