Index

1.Compilazione

La compilazione avviene tramite il comando **make** seguito dal numero della configurazione. Le tre opzioni del MAKEFILE sono:

configurazione 1:

SO_BLOCK_SIZE = 100

SO_REGISTRY_SIZE = 1000

```
1:
    gcc -std=c89 -pthread -g -pedantic -D_GNU_SOURCE -DSO_BLOCK_SIZE=100 -DSO_REGISTRY_SIZE=1000 main.c -lm -o main
```

configurazione 2:

SO_BLOCK_SIZE = 10

SO_REGISTRY_SIZE = 10000

```
2:
    gcc -std=c89 -pthread -g -pedantic -D_GNU_SOURCE -DSO_BLOCK_SIZE=10 -DSO_REGISTRY_SIZE=10000 main.c -o main
```

configurazione 3:

SO_BLOCK_SIZE = 10

SO_REGISTRY_SIZE = 1000

```
3:
    gcc -std=c89 -pthread -g -pedantic -D_GNU_SOURCE -DSO_BLOCK_SIZE=10 -DSO_REGISTRY_SIZE=1000 main.c -o main
```

configurazione custom:

Opzione per scrivere BLOCK_SIZE e REGISTRY_SIZE personalizzati:

```
#setting con entrata libera per block size e registry size
custom:
    @echo -n "SO_BLOCK_SIZE: "
    @read block
    @echo -n "SO_REGISTRY_SIZE: "
    @read registry
    gcc -std=c89 -pthread -g -pedantic -D_GNU_SOURCE -DSO_BLOCK_SIZE = $(block) -DSO_REGISTRY_SIZE = $(registry) main.c -o main

```

significato di ogni flag

- std=c89: Stabilisce il linguagio standard C89.
- pedantic: Disattiva le opzioni del compilatore producendo più errori.

- pthread: Stabilisce il binario per processare threads.
- D_GNU_SOURCE: Abilita estensioni GNU agli standard C e OS supportati dalla libreria GNU C.
- SO_BLOCK_SIZE: La grandezza del blocco nella simulzaione.
- SO_REGISTRY_SIZE: La grandezza massima del libro mastro.

2.Esecuzione

Dopo di aver compilato il programma solo ci manca inizializzarlo. Per questo si puo fare in due maniere diverse: passando un file con tutta la configurazione, o scriverla manualmente.

con file di configurazione

Nel caso d'inviare un file configuration, si passa come argomento di esecuzione.

```
./main conf1.dat
```

scritura manuale

Per scrivere la configurazione manualmente si deve scrivere come secondo argomento la parola "mano" o "manuale".

```
./main manual
```

aggiunge segnali

Nel caso delle segnali per forzare certe transazioni, non è obbligatorio, ma se si aspetta fare questo si aggiunge un terzo argomento con l'indirizzo del file con tutte le transazioni che si aspettano.

```
./main conf1.dat transactions.dat
```

in questo esempio: **transactions.dat** contiene tutte le transazioni.

3.Strutture

Le strutture sono gruppi di variabili che rapressentano un oggetto della vita reale.

Configurazione

Questa struttura solo serve per avere un archivio di dati ordinati dei dati letti del file di configurazione. Questi dati sono:

variables	descripcion
SO_USERS_NUM	numero di processi utente
SO_NODES_NUM	numero di processi nodo
SO_BUDGET_INIT	budget iniziale di ciascun processo utente
SO_REWARD	percentuale di reward pagata da ogni utente per il processo di una transazione
SO_MIN_TRANS_GEN_NSEC	minimo valore del tempo che trascorre fra la generazione di una transazione e la seguente da parte di un utente
SO_MAX_TRANS_GEN_NSEC	massimo valore del tempo che trascorre fra la generazione di una transazione e la seguente da parte di un utente
SO_RETRY	numero massimo di fallimenti consecutivi nella generazione di transazioni dopo cui un processo utente termina
SO_TP_SIZE	numero massimo di transazioni nella transaction pool dei processi nodo
SO_BLOCK_SIZE	numero di transazioni contenute in un blocco
SO_MIN_TRANS_PROC	minimo valore del tempo simulato(nanosecondi) di processamento di un blocco da parte di un nodo
SO_MAX_TRANS_PROC	massimo valore del tempo simulato(nanosecondi) di processamento di un blocco da parte di un nodo
SO_REGISTRY_SIZE	numero massimo di blocchi nel libro mastro.
SO_SIM_SEC	durata della simulazione.
SO_NUM_FRIENDS	numero di nodi amici dei processi nodo(solo per la versione full)
SO_HOPS	numero massimo d'inoltri di una transazione verso nodi amici prima che il master creai un nuovo nodo

```
1
2  /*struttura della configurazione.*/
3  typedef struct Configurazione{
```

```
    int SO_USERS_NUM;           /*numero di processi utente*/
    int SO_NODES_NUM;           /*numero di processi nodo*/
    int SO_BUDGET_INIT;         /*budget iniziale di ciascun processo
utente*/
    int SO_REWARD;              /*la percentuale di reward pagata da ogni
utente per il processamento di una transazione*/
    long int SO_MIN_TRANS_GEN_NSEC; /*minimo valore del tempo che
trascorre fra la generazione di una transazione e la seguente da parte di
un utente*/
    long int SO_MAX_TRANS_GEN_NSEC; /*massimo valore del tempo che
trascorre fra la generazione di una transazione e la seguente da parte di
un utente*/
    int SO_RETRY;               /*numero massimo di fallimenti consecutivi
nella generazione di transazioni dopo cui un processo utente termina*/
    int SO_TP_SIZE;             /*numero massimo di transazioni nella
transaction pool dei processi nodo*/
    long int SO_MIN_TRANS_PROC_NSEC;/*minimo valore del tempo
simulato(nanosecondi) di processamento di un blocco da parte di un nodo*/
    long int SO_MAX_TRANS_PROC_NSEC;/*massimo valore del tempo
simulato(nanosecondi) di processamento di un blocco da parte di un nodo*/
    int SO_SIM_SEC;             /*durata della simulazione*/
    int SO_FRIENDS_NUM;         /*solo per la versione full. numero di nodi
amici dei processi nodo (solo per la versione full)*/
    int SO_HOPS;                /*solo per la versione full. numero massimo
di inoltri di una transazione verso nodi amici prima che il master creai
un nuovo nodo*/
}Configurazione;

extern Configurazione configurazione;

```

Questa struttura è gia dichiarata con la variabile configurazione perchè solo c'è una lettura delle variabili di configurazione.

Lettura Configurazione

Legge File

```
/*Funzione che cerca la maniera di leggere il config file.*/
void readconf(char fileName[]){
    /*secondo lo std c89 tutte le variabile devono
    essere dichiarate prima del primo codice */
    FILE *file= fopen(fileName, "r");

    if(!file){
        printf("non si trova il config file.\n");
        exit(EXIT_FAILURE);
    }else{
        char line[20];/*str per prendere le righe*/

        /*Inserisco le variabili riga per riga alla struttura.*/
        fscanf(file,"%d",&configurazione.SO_USERS_NUM);
        printf("SO_USERS_NUM: %d\n",configurazione.SO_USERS_NUM);
        fscanf(file,"%d",&configurazione.SO_NODES_NUM);
        printf("SO_NODES_NUM: %d\n",configurazione.SO_NODES_NUM);
        fscanf(file,"%d",&configurazione.SO_BUDGET_INIT);
        printf("SO_BUDGET_INIT: %d\n",configurazione.SO_BUDGET_INIT);
```

```
        fscanf(file,"%d",&configurazione.SO_REWARD);
        printf("SO_REWARD: %d\n",configurazione.SO_REWARD);
        fscanf(file,"%ld",&configurazione.SO_MIN_TRANS_GEN_NSEC);
        printf("SO_MIN_TRANS_GEN_NSEC:
%ld\n",configurazione.SO_MIN_TRANS_GEN_NSEC);
        fscanf(file,"%ld",&configurazione.SO_MAX_TRANS_GEN_NSEC);
        printf("SO_MAX_TRANS_GEN_NSEC:
%ld\n",configurazione.SO_MAX_TRANS_GEN_NSEC);
        fscanf(file,"%d",&configurazione.SO_RETRY);
        printf("SO_RETRY: %d\n",configurazione.SO_RETRY);
        fscanf(file,"%d",&configurazione.SO_TP_SIZE);
        printf("SO_TP_SIZE: %d\n",configurazione.SO_TP_SIZE);
        fscanf(file,"%ld",&configurazione.SO_MIN_TRANS_PROC_NSEC);
        printf("SO_MIN_TRANS_PROC_NSEC:
%ld\n",configurazione.SO_MIN_TRANS_PROC_NSEC);
        fscanf(file,"%ld",&configurazione.SO_MAX_TRANS_PROC_NSEC);
        printf("SO_MAX_TRANS_PROC_NSEC:
%ld\n",configurazione.SO_MAX_TRANS_PROC_NSEC);
        fscanf(file,"%d",&configurazione.SO_SIM_SEC);
        printf("SO_SIM_SEC: %d\n",configurazione.SO_SIM_SEC);
        fscanf(file,"%d",&configurazione.SO_FRIENDS_NUM);
        printf("SO_FRIENDS_NUM: %d\n",configurazione.SO_FRIENDS_NUM);
        fscanf(file,"%d",&configurazione.SO_HOPS);
        printf("SO_HOPS: %d\n",configurazione.SO_HOPS);
    }
    fclose(file);/*chiusura del file.*/
}

```

Scrittura Manuale

```
/*scritura manuale dei valori del sistema.*/
void writeConf(){
    printf("inserendo il parametro 'mano' o 'manual' si attiva il
inserimento manuale dei valori\n\n");
    printf("SO_USERS_NUM: ");
    scanf("%d",&configurazione.SO_USERS_NUM);
    printf("SO_NODES_NUM: ");
    scanf("%d",&configurazione.SO_NODES_NUM);
    printf("SO_BUDGET_INIT: ");
    scanf("%d",&configurazione.SO_BUDGET_INIT);
    printf("SO_REWARD: ");
    scanf("%d",&configurazione.SO_REWARD);
    printf("SO_MIN_TRANS_GEN_NSEC: ");
    scanf("%ld",&configurazione.SO_MIN_TRANS_GEN_NSEC);
    printf("SO_MAX_TRANS_GEN_NSEC: ");
    scanf("%ld",&configurazione.SO_MAX_TRANS_GEN_NSEC);
    printf("SO_RETRY: ");
    scanf("%d",&configurazione.SO_RETRY);
    printf("SO_TP_SIZE: ");
    scanf("%d",&configurazione.SO_TP_SIZE);
    printf("SO_MIN_TRANS_PROC_NSEC: ");
    scanf("%ld",&configurazione.SO_MIN_TRANS_PROC_NSEC);
    printf("SO_MAX_TRANS_PROC_NSEC: ");
    scanf("%ld",&configurazione.SO_MAX_TRANS_PROC_NSEC);
    printf("SO_SIM_SEC: ");
    scanf("%d",&configurazione.SO_SIM_SEC);
```

```
    printf("SO_FRIENDS_NUM: ");
    scanf("%d",&configurazione.SO_FRIENDS_NUM);
    printf("SO_HOPS: ");
    scanf("%d",&configurazione.SO_HOPS);

}

```

Transazione

Una transazione è caratterizzata dalle seguenti informazioni:

variabile	descrizione
timestamp	Quando viene effetuata la transazione.
sender	Utente che ha generato la transazione.
receiver	Utente destinatario della somma.
quantita	Quantita di denaro inviata.
reward	denaro dal sender al nodo che processa la transazione

La transazione è inviata dal processo utente che la genera ad uno dei processi nodo, scelto a caso.

```
/*struttura della configurazione.*/
typedef struct Transazione{
    long int timestamp;/*Quando viene effettuata la transazione.*/
    int sender;      /*Utente che ha generato la transazione.*/
    int receiver;    /*Utente destinatario de la somma.*/
    int quantita;    /*Quantita di denaro inviata.*/
    int reward;      /*denaro dal sender al nodo che processa la transazione.*/

}Transazione;

```

printTrans

Uso generico per stampare una transazione. E' usato per le transazioni programate(segnali) e anche quando il processo master invia una transazione a un nuovo nodo creato.

```
void prinTrans(Transazione t){
    printf("%ld: %d -> %d: %d\n",t.timestamp,t.sender,t.receiver,t.quantita);
}

```

RandomInt & RandomLong

Le due funzioni servono per lanciare un numero aleatorio tra min e max. In ogni caso si usano le stesse variabili:

- min: il numero minimo del rango.
- max: il numero massimo del rango.

randomInt serve per semplificare ogni volta che si fa una scelta a caso dentro di ogni thread.
randomlong per ora solo serve per il random sleep.

```
1
2  int randomInt(int min, int max){
3      return rand() % max+min;
4  }
5
6  long randomlong(long int min, long int max){
7      return rand() % max+min;
8  }
9
```

Strutture di tempo

Sezione con tutte le funzione collegate con il timespec o usano un timespec.

getTime

le funzioni di getTime usano il startSimulation come base del tempo durante tutto il processo, e si può chiedere tanti secondi come nanosecondi.

```
1  #define nano 1000000000L
2  extern struct timespec startSimulation;
3
4  /*ritorna il tempo in secondi*/
5  long int getTimeS(){
6      struct timespec now;
7      clock_gettime(CLOCK_REALTIME,&now);
8      return now.tv_sec - startSimulation.tv_sec;
9  }
10
11 /*ritorna il tempo in nanosecondi*/
12 long int getTimeN(){
13     struct timespec now;
14     clock_gettime(CLOCK_REALTIME,&now);
15     return nano*(now.tv_sec-startSimulation.tv_sec) + now.tv_nsec -
   startSimulation.tv_nsec;
16
17 }
```

randomSleep

funzione di nanosleep con un rango tra due numeri:

- min: quantità minima di nanosecondi.
- max: quantità massima di nanosecondi.

```c
/*si ferma per una quantita random di nano secondi*/
void randomSleep(long int min, long int max){

    struct timespec tim;
    tim.tv_sec =0;
    tim.tv_nsec=randomlong(min,max);
    nanosleep(&tim,NULL);

}

```

4.Main

Headers

Basic libraries

```
#include <stdio.h>  /*Standard input-output header*/
#include <stdlib.h> /*Libreria Standard*/
#include <time.h>   /*Acquisizione e manipolazione del tempo*/
#include <stdbool.h>/*Aggiunge i boolean var*/
#include <string.h>/*Standar library for string type*/

```

Specific Libraries

```
#include <unistd.h>      /*Header per sleep()*/
#include <pthread.h>     /*Creazione/Modifica thread*/
#include <semaphore.h>   /*Aggiunge i semafori*/

```

Funzioni Utente

importando le funzioni di User.c sono incluse anche le funzioni di Nodo e Structs.

```
#include "User.c"
#include "print.c"
```

Controllo LIBRO MASTRO

Creazione del Libro_Mastro e Variabili:

- **libroluck**: Semaforo per accedere alla scrittura del libroMastro.
- **libroCounter**: Contatore della quantità di blocchi scritti nel libroMastro.

```
Transazione libroMastro[SO_REGISTRY_SIZE * SO_BLOCK_SIZE];/*libro mastro
dove si scrivono tutte le transazioni.*/
int libroCounter=0;/*Counter controlla la quantitta di blocchi*/
sem_t libroluck;/*Luchetto per accedere solo a un nodo alla volta*/
bool gestoreOccupato;

```

Memoria condivisa

In base a un grupo di variabili condivise si stabilisce un sistema di comunicazione tra i diversi processi. Questi dati condivisi servono ad altri processi in qualche momento, o sono dati servono al main per stampare lo stato dei processi.

Lista Dati Condivisi tra i threads:

```
/*variabili condivise tra diversi thread.*/
int *budgetlist;     /*un registro del budget di ogni utente*/
bool *checkUser;     /*mostra lo stato di ogni utente.*/
sem_t UserStartSem;  /*un semaforo dedicato unicamente per iniziare processi utente*/

int *rewardlist;     /*un registro pubblico del reward totale di ogni nodo.*/
int *poolsizelist;   /*un registro del dimensioni occupate pool transaction*/
sem_t *semafori;     /*semafori per accedere/bloccare un nodo*/
sem_t NodeStartSem;  /*un semaforo dedicato unicamente per iniziare processi nodo*/
Transazione *mailbox;/*struttura per condividere */
bool *checkNode;     /*lista che mostra i nodi che sono attivi.*/

Transazione mainMailbox;
struct timespec startSimulation;

pthread_t *utenti_threads;     /*lista id di processi utenti*/
pthread_t *nodi_threads;     /*lista id di processi nodi  */
Configurazione configurazione;

```

Transazioni programmate

Le transazioni programmate sono una lista di transazioni che vengono letti da un file che contiene una transazione per ogni riga. Ogni riga contiene lo timestamp ,sender, reciever e quantità della transazione. Quando è il momento del timeStamp della transazione viene creata una segnale dal main per forzare che l'utente sender faccia questa transazione.

Lettura del file di transazioni pianificati

questa funzione non ha bisogno di ritornare un array perche può essere passato come parametro della funzione e si scrive direttamente nell'array.
Per questo motivo il return della funzione ritornerà un valore intero che rappresenta la quantità di transazioni programmate.

```

/*legge le transazioni e gli scrive in un array di transazioni per scriverle
dopo nel libro mastro.*/
int leggeLibroDiTransazioni(char fileName[], Transazione programmate[100])
{
    int i = 0;
    FILE *file = fopen(fileName,"r");
    if(!file){
        printf("non si trova il libro di transazioni programmate.\n");
    }else{
        /*legge riga a riga fino alla fine(EOF), mettendo tutti le variabili nell'array

```

```
        delle transazioni programmate.*/
        while(fscanf(file,"%ld %d %d %d",&programmate[i].timestamp,&programmate[i].sender,&programmate[i].receiver,&programmate[i].quantita) != EOF && i<100){
            programmate[i].reward = programmate[i].quantita * configurazione.SO_REWARD / 100;
            if(programmate[i].reward < 1){
                programmate[i].reward =1;
            }
            i++;
        }
    }
    return i;
}

```

Segnale

La segnale è una maniera di forzare un'utente a fare una transazione già creata dal master con valori predefiniti.

```

/*segnale che forza una transazione di un'utente.*/
void segnale(Transazione programmato){
    mailbox[nodoLibero(programmato.sender)] = programmato;/*assegno la transazione in un mailbox*/

    budgetlist[programmato.sender] -= programmato.quantita;
    printf("Segnale ->");
    prinTrans(programmato);
}
```

Nuovo nodo

Funzione che ridimensiona tutte le liste per dopo creare un nuovo nodo e inviare la transazione che non è stata possibile condividere con nessun altro nodo.

```

void* gestore(){
    int i;
    int semvalue;

    while(getTimeS() < configurazione.SO_SIM_SEC){
        if(gestoreOccupato){
            /*resize each list with realloc*/
            poolsizelist=realloc(poolsizelist, (configurazione.SO_NODES_NUM+1)*sizeof(int));
            rewardlist  =realloc(rewardlist , (configurazione.SO_NODES_NUM+1)*sizeof(int));
            semafori     =realloc(semafori, (configurazione.SO_NODES_NUM+1)*sizeof(sem_t));
            checkNode    =realloc(checkNode, (configurazione.SO_NODES_NUM+1)*sizeof(bool));
            nodi_threads=realloc(nodi_threads, (configurazione.SO_NODES_NUM+1)*sizeof(pthread_t));
```

```
            mailbox=realloc(mailbox,
(configurazione.SO_NODES_NUM+1)*6*sizeof(int));

            rewardlist[configurazione.SO_NODES_NUM]=0;
            poolsizelist[configurazione.SO_NODES_NUM]=0;

            /*inizia il nuovo trhead*/

 pthread_create(&nodi_threads[configurazione.SO_NODES_NUM],NULL,nodo,NULL)
;

            mailbox[configurazione.SO_NODES_NUM] = mainMailbox;

            /*si reapre il gestore di nuovi nodi*/
            gestoreOccupato=false;
            configurazione.SO_NODES_NUM++;
        }
        randomSleep(10,10);

    }
}

```

Funzione Master

E' il metodo principale del progetto. Il suoi compiti sono:

- leggere la configurazione, sia file o manuale
- inizzializare tutta la memoria condivisa
- creare tutti i processi nodo e utente
- stampare l'informazione dei processi attivi
- creare un nodo nuovo quando nessun nodo riesce a prendere una transazione dopo HOPS volte.
- chiudere tutti i processi

```
int main(int argc,char *argv[]){
    int i;
    struct timespec now;
    pthread_t thrGestore;


    /*variabili delle transazioni programmate*/
    int programmateCounter;
    bool *programmateChecklist;
    Transazione programmate[100];


    srand(time(0)); /*aleatorio*/

    if(argc<2){
        printf("si aspettava un file con la configurazione o il commando
'manual'.\n");
        exit(EXIT_FAILURE);
    }else if(argc>3){
        printf("troppi argomenti.\n");
        exit(EXIT_FAILURE);
```

```
    }else{
        /*in caso di voler inserire i valori a mano*/
        if( strcmp(argv[1],"mano")==0 || strcmp(argv[1],"manual")==0 ){
            writeConf();
        }else{
            readconf(argv[1]);/*lettura del file*/
        }

        /*lettura di transazioni programmate*/
        if(argc == 3){
            programmateCounter = leggeLibroDiTransazioni(argv[2],
programmate);
            programmateChecklist = malloc(programmateCounter *
sizeof(bool));
            for(i=0; i < programmateCounter; i++){
                programmateChecklist[i] = true;
            }
        }else{
            programmateCounter = 0;
        }


        /*now that we have all the variables we can start the process
        master*/
        sem_init(&libroluck,configurazione.SO_NODES_NUM,1);/*inizia il
semaforo del libromastro*/

 /*sem_init(&mainSem,configurazione.SO_NODES_NUM+configurazione.SO_USERS_
NUM,1);*/
        gestoreOccupato=false;
        clock_gettime(CLOCK_REALTIME,&startSimulation);

        /*generatore dei nodi*/
        poolsizelist=calloc(configurazione.SO_NODES_NUM , sizeof(int));
        rewardlist=calloc(configurazione.SO_NODES_NUM , sizeof(int));
        semafori=calloc(configurazione.SO_NODES_NUM , sizeof(sem_t));
        mailbox=calloc(configurazione.SO_NODES_NUM , 6 * sizeof(int));
        nodi_threads = malloc(configurazione.SO_NODES_NUM *
sizeof(pthread_t));
        checkNode = calloc(configurazione.SO_NODES_NUM , sizeof(bool));
        for(i=0;i<configurazione.SO_NODES_NUM;i++){
            pthread_create(&nodi_threads[i],NULL,nodo,NULL);
        }

        pthread_create(&thrGestore,NULL,gestore,NULL);

        /*generatore dei utenti*/
        budgetlist=calloc(configurazione.SO_USERS_NUM , sizeof(int));
        utenti_threads = calloc(configurazione.SO_USERS_NUM ,
sizeof(pthread_t));
        checkUser = calloc(configurazione.SO_USERS_NUM , sizeof(bool));
        for(i=0;i<configurazione.SO_USERS_NUM;i++){
            pthread_create(&utenti_threads[i],NULL,utente,NULL);
        }


        while(getTimeS() < configurazione.SO_SIM_SEC){

```

```
            sleep(1);
            clear();

            /*show last update*/
            printf("ultimo aggiornamento:
%ld/%d\n",getTimeS(),configurazione.SO_SIM_SEC);


            if(libroCounter > SO_REGISTRY_SIZE){
                break;
            }


            if(!printStatus(40)){
                printf("tutti gli utenti sono disattivati");
                break;
            }

            /* transazioni programmate mancanti*/
            for(i=0; i< programmateCounter; i++){
                if(programmate[i].timestamp <= getTimeN() &&
programmateChecklist[i]){
                    segnale(programmate[i]);
                    programmateChecklist[i] = false;
                }
            }

        }
        finalprint();

        /*kill all the threads*/
        /*for(i=0; i<configurazione.SO_NODES_NUM ; i++){
            pthread_cancel(nodi_threads[i]);
        }
        for(i=0; i<configurazione.SO_USERS_NUM; i++){
            pthread_cancel(utenti_threads[i]);
        }*/
    }
    return 0;
}
```

5.Nodo

Importa Variabili Globali

Importa funzioni e strutture di Structs.

```
#include "Structs.c"
#define defaultSender -1
```

Controllo del LIBRO_MASTRO

Import del libroMastro e tutte le variabili:

- **libroluck**: Semaforo per accedere alla scrittura del libroMastro.
- **libroCounter**: Contatore che indica la quantità di blocchi scritti nel libroMastro.

```
extern Transazione libroMastro[SO_REGISTRY_SIZE * SO_BLOCK_SIZE];/*libro mastro dove si scrivono tutte le transazioni.*/
extern int libroCounter;/*Counter controlla la quantitta di blocchi*/
extern sem_t libroluck; /*luchetto per accedere solo un nodo alla volta*/

```

Memoria Condivisa

Il nodo non ha bisogno delle variabili degli utenti. Quindi solo servono le variabili del main e dei altri nodi nodi.

```
/*variabili condivise tra diversi thread.*/
extern int *rewardlist;     /*un registro publico del reward totale di ogni nodo.*/
extern sem_t *semafori;     /*semafori per accedere/bloccare un nodo*/
extern Transazione *mailbox;/*struttura per condividere */
extern int *poolsizelist;   /*un registro del dimensioni occupate pool transaction*/
extern bool *checkNode;
extern pthread_t *nodi_threads;

extern Transazione mainMailbox;
extern bool gestoreOccupato;


extern Configurazione configurazione;

```

trova ID del Nodo

Per colpa del pedantic nel Makefile non possiamo fare un cast da integer a un puntatore void. Questo ci limita per passare argomenti a un thread, e per tanto anche ci impide passare l'ID al nodo come un argomento. Per questo motivo dobbiamo creare una funzione che trova l'ID del nodo in base alla posizione del thread nella lista nodi_threads. A differenza del trovaUtenteID, questa funzione inizia la ricerca da SO_NODES_NUM, lo facciamo per ridurre la quantità di cicli che fanno i nodi creati a metà simulazione da parte del main.

```c
/*cerca la posizione del thread del nodo.*/
int trovaNodoID(){
    int id;

    for(id=configurazione.SO_NODES_NUM; id>=0; id--){
        if(pthread_self() == nodi_threads[id]){
            break;
        }
    }
    return id;
}
```

transazione di riassunto

Questo metodo genera l'ultima transazione del blocco.
Questa transazione fa un riassunto di tutto quello che ha guadagnato il nodo in questo blocco.

```c

/*funzione dell'ultima transazione del blocco.*/
Transazione riasunto(int id, int somma){
    Transazione transaction;
    transaction.sender    = defaultSender;
    transaction.receiver  = id; /*id del nodo*/
    transaction.quantita  = somma; /*la somma di tutto il reward generato*/
    transaction.timestamp = getTimeN();/*quanto tempo ha passato dal inizio della simulazione.*/
    return transaction;
}

```

Invia transazione a nodo amico

Questa funzione invia una transazione a un amico o crea un nuovo nodo per inviarselo.

```c
void inviaAdAmico(int *amici,int id){
    bool inviaAmico=true;
    int hops=0;
    int i;
    int len = sizeof(amici)/sizeof(int);
    while(inviaAmico){
        for(i=0; i<len && inviaAmico;i++){
            if(checkNode[amici[i]]){/*evito inviare a un nodo pieno.*/
                if(sem_trywait(&semafori[*(amici+i)])){
                    mailbox[amici[i]]=mailbox[id];
                    inviaAmico=false;
                }
            }
        }
        if(inviaAmico){
            /*printf("Il nodo %d non ha nessun amico\n",id);*/
            hops++;
            if(hops > configurazione.SO_HOPS){
                if(!gestoreOccupato){
                    gestoreOccupato=true;
```

```
                    mainMailbox=mailbox[id];
                    amici = realloc(amici,(len+1)*sizeof(int));
                    amici[len]= configurazione.SO_NODES_NUM;
                    hops=0;
                }
                inviaAmico=false;
            }
        }
    }
    sem_post(&semafori[id]);
}
```

Funzione principale del nodo.

```
void* nodo(){
    /*creazioni dei dati del nodo*/
    int id = trovaNodoID();
    int i;
    int hops=0;
    int counterBlock=0;/*contatore della quantita di transazioni nel blocco*/
    int sommaBlocco=0; /*somma delle transazioni del blocco atuale*/
    Transazione blocco[SO_BLOCK_SIZE];
    Transazione pool[1000];/*stabilisce 1000 come la grandezza massima del pool, cmq si ferma in configurazione.SO_TP_SIZE*/
    Transazione finalReward;
    int mythr;
    int semvalue;/*valore del semaforo*/
    int *amici = calloc(configurazione.SO_FRIENDS_NUM, sizeof(int));
    bool inviaAmico=true;
    for(i=0;i<configurazione.SO_FRIENDS_NUM;i++){
        do{
            amici[i] = randomInt(0,configurazione.SO_NODES_NUM);
        }while(amici[i]==id);
    }
    sem_init(&semafori[id],configurazione.SO_USERS_NUM,1);/*inizia il semaforo in 1*/
    rewardlist[id]=0;/*set il reward di questo nodo in 0*/
    poolsizelist[id]=0;/*set full space available*/
    checkNode[id] = true;
    /*mythr = pthread_self();
    /*printf("Nodo #%d creato nel thread %d\n",id,mythr);*/

    /*inizio del funzionamento*/
    while(checkNode[id]){

        /*aggiorno il valore del semaforo*/
        sem_getvalue(&semafori[id],&semvalue);
        if(semvalue <= 0){
            /*printf("hay algo en el mailbox #%d\n",id);*/
            /*scrivo la nuova transazione nel blocco e nella pool*/
            if(counterBlock==SO_BLOCK_SIZE/2 && inviaAmico){
                inviaAdAmico(amici,id);
                inviaAmico=false;
                continue;
             }
             pool[poolsizelist[id]]=mailbox[id];
```

```
41              blocco[counterBlock]=mailbox[id];
42
43              /*somma il reward*/
44              sommaBlocco    += blocco[counterBlock].reward;
45              rewardlist[id] += blocco[counterBlock].reward;/*si mette al
registro publico totale*/
46
47              /*incremento i contatori di posizione di pool e block*/
48              counterBlock++;
49              poolsizelist[id]++;
50
51              if(counterBlock == SO_BLOCK_SIZE - 1){
52                 /*si aggiunge una nuova transazione come chiusura del
blocco*/
53                 blocco[counterBlock]=riasunto(id, sommaBlocco);/*aggiunge
la transazione al blocco.*/
54
55                 sem_wait(&libroluck);
56                 for(i=0;i< SO_BLOCK_SIZE;i++){
57                     libroMastro[(libroCounter * SO_BLOCK_SIZE) + i] =
blocco[i];
58                     /*se hai bisogno di dimostrare che si scrive il libro
mastro,
59                     scomenta il seguente print. Questo stampa tutto il
blocco quando si
60                     scrive nel libroMastro.*/
61                     /*prinTrans(blocco[i]);*/
62                 }
63                 /*si spostano i contatori*/
64                 libroCounter++;
65                 sem_post(&libroluck);
66                 counterBlock=0;
67                 sommaBlocco=0;
68                 inviaAmico=true;
69
 randomSleep(configurazione.SO_MIN_TRANS_PROC_NSEC,configurazione.SO_MAX_T
RANS_PROC_NSEC);
70                 /*free(&mailbox[id]);*/
71
72
73             }
74             sem_post(&semafori[id]);/*stabilisco il semaforo come di nuovo
disponibile*/
75             if(poolsizelist[id] >= configurazione.SO_TP_SIZE){
76                 checkNode[id]=false;
77             }
78
79         }
80
81     }
82     /*nodo zombie*/
83     while(getTimeS()<configurazione.SO_SIM_SEC){
84         sem_getvalue(&semafori[id],&semvalue);
85         if(semvalue <= 0){
86             inviaAdAmico(amici,id);
87         }
88     }
89 }
```

6.Utente

import

S'importa il codice di Node.c che importa tutte le funzioni di Node e Structs.

```
#include "Node.c"

```

Importa Variabili Globali

Controllo del Libro_Mastro

Importazione del libroMastro e tutte le variabili:

- **libroluck**: Semaforo per accedere alla scrittura del libroMastro.
- **libroCounter**:Contatore che indica la quantità di blocchi scritti nel libroMastro.

```
extern Transazione libroMastro[SO_REGISTRY_SIZE * SO_BLOCK_SIZE];/*libro mastro dove si scrivono tutte le transazioni.*/
extern int libroCounter;/*Counter controlla la quantitta di blocchi*/
extern sem_t libroluck;/*luchetto per accedere solo un nodo alla volta*/

```

Sincronizzazione tra Processi

Importa tutte le variabili condivise

```
/*variabili condivise tra diversi thread.*/
extern int *budgetlist;     /*un registro del budget di ogni utente*/
extern bool *checkUser;

extern int *rewardlist;     /*un registro publico del reward totale di ogni nodo.*/
extern int *poolsizelist;  /*un registro del dimensioni occupate pool transaction*/
extern sem_t *semafori;     /*semafori per accedere/bloccare un nodo*/
extern Transazione *mailbox;/*struttura per condividere */

extern Configurazione configurazione;
extern pthread_t *utenti_threads;      /*lista id dei processi utenti*/

```

trova ID del utente

Per colpa del pedantic nel Makefile non possiamo fare un cast da integer a un puntatore void. Questo ci limita per passare argomenti a un thread, e per tanto anche ci impide passare l'ID all'utente come un argomento. Per questo motivo dobbiamo creare una funzione che trova l'ID dell'utente in base alla posizione del thread nella lista utenti_threads.

```c
/*cerca la posizione del thread del utente.*/
int trovaUtenteID(){
    int id;
    for(id=0;id<configurazione.SO_USERS_NUM; id++){
        if(pthread_self() == utenti_threads[id]){
            break;
        }
    }
    return id;
}
```

Aggiornamento Libro_Mastro

L'aggiornamento tramite Libro_Mastro avviene tramite una sola funzione.

```c
/*aggiornamento del budget in base al libro.*/
int userUpdate(int id, int lastUpdate){
    int i;
    while(lastUpdate < libroCounter){
        for(i=lastUpdate*SO_BLOCK_SIZE; i < (lastUpdate+1)*SO_BLOCK_SIZE; i++){
            if(libroMastro[i].receiver == id && libroMastro[i].sender != -1){
                budgetlist[id] += libroMastro[i].quantita - libroMastro[i].reward;
            }
        }
        lastUpdate++;
    }
    return lastUpdate;
}

```

trova nodo

serve per trovare un nodo libero per fare la transazione.

```c
/*cerca un nodo libero per fare la trasazione.*/
int nodoLibero(int id){
    int nodo;
    int retry = 0;
    do{
        nodo = randomInt(0,configurazione.SO_NODES_NUM);
        if( retry > configurazione.SO_RETRY){
            /*printf("L'utenete %d non ha trovato nessun nodo libero\n",id);*/
            checkUser[id]= false;
            pthread_cancel(utenti_threads[id]);
        }
        retry++;
    }while(sem_trywait(&semafori[nodo])<0 && checkUser[id]);
    return nodo;

}

```

Generatore di Transazione

comprime tutto il processo di generare la transazione.

```
Transazione generateTransaction(int id){
    int altroUtente;
    Transazione transaccion;
    transaccion.sender   = id;
    transaccion.quantita = randomInt(2,budgetlist[id]/2);/*set quantita a caso*/
    transaccion.reward   = transaccion.quantita * configurazione.SO_REWARD/100;/*percentuale de la quantita*/

    /*se il reward non arriva a 1, allora diventa 1*/
    if(transaccion.reward < 1){
        transaccion.reward = 1;
    }

    /*ricerca del riceiver*/
    /*debo reparar lo de los intentos*/
    do{
        altroUtente= randomInt(0,configurazione.SO_USERS_NUM);
    }while(altroUtente==id || !checkUser[altroUtente]);
    transaccion.receiver = altroUtente;
    /*calcola il timestamp in base al tempo di simulazione.*/
    transaccion.timestamp = getTimeN();

    return transaccion;
}

```

Processo Utente Principale

```
/*PROCESSO UTENTE:*/
void* utente(){
    int id = trovaUtenteID();                       /*Id processo utente*/
    int i;
    pthread_t mythr = pthread_self();          /*Pid thread processo utente*/
    int lastUpdate = 0;                        /*questo controlla l'ultima versione del libro mastro*/
    int retry=0;
    /*setting default values delle variabili condivise*/
    checkUser[id] = true;
    budgetlist[id] = configurazione.SO_BUDGET_INIT;

    /*printf("Utente #%d creato nel thread %d\n",id,mythr);*/

    while(checkUser[id]){

        lastUpdate = userUpdate(id,lastUpdate);  /*Aggiorniamo Budgetdel Processo Utente*/

        if(budgetlist[id]>=2){                    /*Condizione Budget >= 2*/

```

```
            Transazione transaction;              /*Creiamo una nuova transazione*/
            retry = 0;
            transaction = generateTransaction(id);/*Chiamiamo la func generateTransaction*/

            /*scelglie un nodo libero a caso*/
            mailbox[nodoLibero(id)] = transaction;
            budgetlist[id] -= transaction.quantita;
        }else{
            retry++;
        }

        randomSleep( configurazione.SO_MIN_TRANS_GEN_NSEC , configurazione.SO_MAX_TRANS_GEN_NSEC);

        if(retry >= configurazione.SO_RETRY){/*Se raggiunge il n° max di tentativi*/
            /*printf("utente %d fermato\n",id);       /*ferma il procceso*/
            checkUser[id]=false;
        }
    }
}
```

7.Prints

Questa sezione contiene tutto il codice che collega con le funzioni che servono per stampare le funzioni.
Le principali funzioni sono:

printStatus: mostra una tabella con la informazione degli utenti piu attivi e i nodi. Si usa per mostrare i dati aggiornati in ogni secodo de la simulazione.

finalPrint: mostra una tabella con tutti gli utenti ,utti i nodi, e più dati, come il nome indica, si usa per stampare tutti i dati alla fine della simulazione.

Macros

Lista di macros che ci servono per stampare tutti i valori:

- **clear**: pulisce lo schermo.
- **MAX**: ritorna il numero maggiore tra i due.
- **MIN**: invia il numero minore tra i due.
- **boolString**: fa la funzione di %b in altri linguagi di programmmazione.

```
/*macros per il print*/
#define clear() printf("\033[H\033[J") /*clear the screen*/
#define MAX(x,y) ((x>y)?x:y) /*max between to parameters*/
#define MIN(z,w) ((z<w)?z:w) /*min between to parameters*/
#define boolString(b) ((b) ? "True":"False")/*make the %b*/

```

memoria condivise

Tutte le variabili che devono stampare le funzioni di stampa

```
/*variabili degli utenti*/
extern int *budgetlist;
extern bool *checkUser;

/*variabili degli utenti*/
extern int *rewardlist;
extern int *rewardlist;
extern int *poolsizelist;
extern bool *checkNode;

```

Compare Function

Metodo che compara due valori e restituisce un numero positivo, se b è piu grande di a ,e negativo, se b è piu piccolo di a.

```
int cmpfunc(const void  *a, const void *b) {
    return(budgetlist[*((int*)b)]-budgetlist[*((int*)a)]);
}
```

Sort risults

Metodo di ordinamento dei processi in modo decrescente (dal piu grande al piu piccolo).

```
int * sort(){
    int dim=configurazione.SO_USERS_NUM;
    int *r=malloc(sizeof(int)*dim);
    int i;
    for(i=0; i<dim; i++)
        r[i]=i;

    qsort(r, dim, sizeof(int), cmpfunc);
    return r;
}
```

PrintStatus Nodes and Users

Questo metodo non solo mostra lo stato di tutti gli utenti e i nodi, ritorna anche una variabile boolean per identificare se ci sono ancora utenti disponibili.

```
bool printStatus(int nstamp){
    /*User var*/
    int activeUsers=0;
    int inactiveUsers=0;
    int sommaBudget=0;
    bool ActiveU;
    /*Node var*/
    int activeNodes=0;
    int inactiveNodes=0;
    int sommaRewards=0;
    bool ActiveN;
    /*Share var*/
    int i=0;
    int *pa;
    int
dim=MIN(MAX(configurazione.SO_USERS_NUM,configurazione.SO_NODES_NUM),
nstamp);
    pa=sort();
    printf("\n\n");
    printf("----------------------------------------------------------------
----------\n");
    printf("||  User_ID |  Budget  |  Status |##|  Node_ID  |  Rewards  |
Status ||\n");

 printf("||==============================|##|============================
=====||\n");

    /*Stampa risultati*/
    for(i=0;
i<MAX(configurazione.SO_USERS_NUM,configurazione.SO_NODES_NUM); i++){
        if(i<configurazione.SO_USERS_NUM){
            sommaBudget += budgetlist[*(pa+i)];
            checkUser[*(pa+i)] ? activeUsers++ : inactiveUsers++;
            if(i<dim){
                printf("||%10d|%10d|%9s|#",pa[i],budgetlist[*(pa+i)],
boolString(checkUser[*(pa+i)]));
```

```
            }
        }else if(i<dim){
            printf("||           |           |          |#");
        }


        if(i < configurazione.SO_NODES_NUM){
            sommaRewards+=rewardlist[i];
            checkNode[i] ? activeNodes++ : inactiveNodes++;
            if(i<dim){
                printf("#|%11d|%11d|%9s||\n", i,
rewardlist[i],boolString(checkNode[i]));
            }

        }else if(i<dim){
            printf("#|           |           |          ||\n");
        }
    }

    printf("-----------------------------------------------------------------
----------\n");
    printf("||  Active Users  |  Tot Budget  |##|  Active Nodes  |   Tot
Rewards  ||\n");

 printf("||%16d|%14d|##|%16d|%16d||\n",activeUsers,sommaBudget,activeNodes
, sommaRewards);
    printf("\n");

    return activeUsers>0;
}

```

final print

Questo metodo fa l'utima stampa del proggetto. Mostrando tutti gli utenti e mostrando anche la grandezza de la Transaction Pool. Serve come riassunto della simulazione.

```
void finalprint(){
    /*User var*/
    int activeUsers=0;
    int inactiveUsers=0;
    int sommaBudget=0;
    bool ActiveU;
    /*Node var*/
    int activeNodes=0;
    int inactiveNodes=0;
    int sommaRewards=0;
    bool ActiveN;
    /*Share var*/
    int i=0;
    int dim = MAX(configurazione.SO_USERS_NUM,
configurazione.SO_NODES_NUM);

    printf("\n\n");
    printf("-----------------------------------------------------------------
------------------\n");

```

```
    printf("||  User_ID |  Budget  |  Status |##|  Node_ID  |  Rewards  |
p_size | Status ||\n");

 printf("||================================|##|============================
==============||\n");
    for(i=0; i< dim; i++){
        if( i < configurazione.SO_USERS_NUM){
            sommaBudget += budgetlist[i];

            checkUser[i] ? activeUsers++ : inactiveUsers++;

            printf("||%10d|%10d|%9s|#",i,budgetlist[i],
boolString(checkUser[i]));
        }else{
            printf("||          |          |         |#");
        }

        if(i< configurazione.SO_NODES_NUM){
            sommaRewards+=rewardlist[i];

            checkNode[i] ? activeNodes++ : inactiveNodes++;
            printf("#|%11d|%11d|%9d|%8s||\n", i,
rewardlist[i],poolsizelist[i],boolString(checkNode[i]));
        }else{
            printf("#|           |           |         |        ||\n");
        }
    }
    printf("-------------------------------------------------------------
-------------------\n");
    printf("||   Active Users  |   Inactive Users  |##|   Active Nodes  |
Inactive Nodes  ||\n");
    printf("||%17d|%19d|##|%17d|%18d||\n",activeUsers,inactiveUsers,
activeNodes, inactiveNodes);
    printf("||-------------------------------------------------------------
-----------------||\n");
    printf("||    Tot Rewards  |%59d||\n",sommaRewards);
    printf("||-------------------------------------------------------------
-----------------||\n");
    printf("||    Tot Budgets  |%59d||\n",sommaBudget);
    printf("||-------------------------------------------------------------
-----------------||\n");
    printf("||    Tot Block    |%59d||\n", libroCounter);
    printf("-------------------------------------------------------------
-------------------\n");

    /*motivo del termine*/
    if(activeUsers==0){
        printf("Motivo di chiusura:tutti gli utenti sono disattivati.\n");
    }else if(libroCounter >= SO_REGISTRY_SIZE){
        printf("Motivo di chiusura: libroMastro pieno.\n");
    }else{
        printf("Simulazione finita perfettamente.\n");
    }
}
```